Universidad Politécnica de Madrid

Escuela Universitaria de Ingeniería Técnica de Telecomunicaciones

PROYECTO FIN DE CARRERA

REDES VSAT CON EL ESTÁNDAR DVB-S

Álvaro Crespo Matamala

Narciso Covacho Pedraz

Tutor: Francisco José Arqués Orobón

Dpto. Ingeniería Audiovisual y Comunicaciones

Julio 2014

“La mayor recompensa al esfuerzo de un hombre no es lo que obtiene a cambio, sino en la persona en la que se convierte al hacerlo”

John Ruskin

Agradecimientos

Llego ese día que parecía que no iba a llegar, un día que nunca aparecía en el calendario pero al que nos acercábamos poco a poco, el día que pondríamos fin a esta etapa de nuestra vida. Durante todo este tiempo hemos crecido tanto profesional como personalmente gracias a las personas que nos han acompañado por este camino, sin ellos todo esto no hubiera sido posible.

En primer lugar queremos agradecer a nuestro tutor José Arqués por toda por la paciencia demostrada con nosotros, pero sobre por su forma de ser que ha hecho que le veamos no solo como un profesor, sino como un compañero del que nos llevamos muy buenos recuerdos.

En segundo lugar dar las gracias a todos vosotros que habéis hecho que estos años hayan sido increíbles y difíciles de olvidar. Mención especial para aquel grupo que formamos el primer año, que pese al paso del tiempo y a las derivas de la vida, hemos permanecido juntos y apoyándonos.

También dar las gracias a Javier de la Confederación Hidrográfica del Tajo por la ayuda prestada en la realización de las pruebas.

Álvaro

A mi familia, por el apoyo recibido desde que empecé la carrera y los consejos que me han dado durante su transcurso que han esto posible. A mi novia Patricia por saber estar ahí cuando más lo necesitaba sacándome una sonrisa en los momentos más difíciles y por supuesto a mi compañero Narciso Covacho, que se ha convirtiendo en un gran amigo desde que nos conocimos el primer año. ¡Narzo esto está hecho!

Narciso

A mis padres, por todo su sacrificio y lucha en que mi única preocupación fueran los estudios, por hacerme creer que era capaz de todo si me lo proponía y por inculcarme unos valores que me han convertido en la persona que soy. Sin olvidar a mi compañero y amigo Álvaro Crespo, que ha sido un gran compañero de viaje no solo en este trabajo, sino en toda la carrera. Lo hemos conseguido.

¡Gracias a todos!

RESUMEN

Actualmente las redes VSAT (Very Small Aperture Terminal) están adquiriendo una mayor importancia en las comunicaciones por satélite debido a las nuevas aplicaciones que se están desarrollando tanto a nivel empresarial como a nivel de usuario final. El presente proyecto pretende hacer un estudio de este tipo de red para presentarla como una solución al problema de querer conectar estaciones dispersas, que por el perfil del terreno hace difícil la conexión de las mismas a través de las redes terrestres convencionales.

Los nuevos estándares están haciendo que este tipo de redes proliferen muy deprisa ya que se consigue una mayor flexibilidad que con los estándares precedentes para este tipo de red. En concreto, en este proyecto se ha estudiado el estándar abierto DVB-S desarrollado por el grupo de trabajo DVB por ser uno de los más aceptado internacionalmente.

Para comprender este sistema de comunicaciones, el proyecto está estructurado en dos partes.

En la primera parte se hace una revisión de cómo han evolucionado las comunicaciones satelitales VSAT, indicando las ventajas y desventajas de su implementación y sobre todo la orientación que éstas muestran a la utilización de los estándares DVB. Posteriormente se realiza un estudio de los estándares DVB-S y DVB-RCS en donde se profundiza en conceptos claves tales como el Multiplexado de Transporte MPEG-2, los mecanismos de envío de mensajes de señalización, etc.

En la segunda parte del proyecto se presta atención a la seguridad de la red, analizando los mecanismos propios que presenta el estándar DVB así como los diferentes protocolos de seguridad existentes en las capas superiores para una protección adicional.

Para concluir el proyecto se han creado dos aplicaciones, la primera como método didáctico para comprender mejor el comportamiento de las redes VSAT con el estándar DVB-S, y una segunda aplicación con carácter comercial para la transferencia

de ficheros de manera segura con características específicas, enfocada particularmente en redes VSAT, aunque siendo posible su uso en otras redes.

ABSTRACT

Nowadays VSAT networks (Very Small Aperture Terminal) are becoming more important in satellite communications, due to several new applications that are being developed both at company level and end user level. This project aims to make a study of this type of network to present it as a solution to the problem of wanting to connect scattered stations, because the terrain profile makes difficult to connect them via conventional terrestrial networks.

New standards are making that such networks proliferate very quickly for the reason that a more flexibility than the previous standards for this type of network is achieved. Specifically, this project has studied the open standard DVB-S developed by the DVB workgroup as one of the most internationally accepted.

To understand this communication system, this project is structured in two different parts:

On one hand, in the first part a review about how VSAT satellite communications have evolved, indicating the advantages and disadvantages of its implementation and above all, the guidance that they show to the use of the DVB standards. Subsequently, a study of the DVB-S and DVB-RCS standards is developed, where delves into key concepts such as MPEG-2 Multiplexed Transport, mechanisms of transmission of signaling messages, etc.

On the other hand, in the second part of the project, we focus on network security, analyzing the mechanisms presented by the DVB standard and various existing security protocols in the upper layers for an extra protection.

To complete the project two different applications have been developed: the first one as a teaching method to better understand the behavior of VSAT networks in DVB-S standard, and the second one with a commercial basis for transferring files securely with specific features applications focused particularly in VSAT networks, although with a possible use on other networks.

Índice de contenidos

1. INTRODUCCIÓN 1

2. COMUNICACIONES POR SATÉLITE BASADAS EN LAS REDES VSAT 3

2.1. CONFIGURACIÓN DE LAS REDES VSAT 5

2.1.1. Red en malla 5

2.1.2. Red en estrella 7

2.1.3. Red hibrida 8

2.2. CLASIFICACIÓN DE LAS REDES VSAT 8

2.2.1. Redes unidireccionales (redes de difusión) 8

2.2.2. Redes bidireccionales (redes interactivas) 9

2.2.3. Redes corporativas 11

2.3. ESTACIONES TERRESTRES DE LAS REDES VSAT 11

2.3.1. Estación VSAT 11

2.3.2. Estación HUB 14

2.4. TCP SOBRE ENLACES SATELITALES 17

2.5. ESTÁNDARES PARA COMUNICACIONES POR SATÉLITE 20

3. EL PROYECTO DVB 25

3.1. ¿QUÉ ES EL DVB? 25

3.2. EL SISTEMA MPEG-2 27

3.2.1. Formación de los flujos de señal MPEG-2 27

3.2.2. Formación del flujo de transporte TS 29

3.2.2.1. Unidades básicas 29

3.2.2.2. Empaquetado PES 30

3.2.2.3. Multiplexación de los paquetes PES 32

3.2.3. Información específica de los programas (PSI) 36

3.3. DVB-S (*DIGITAL VIDEO BROADCASTING BY SATELLITE*) 36

3.3.1. Sistema de transmisión DVB-S 37

3.3.1.1. Definición del sistema 37

3.3.1.2. Parámetros del sistema 38

3.3.1.3. Codificación de canal 40

3.3.2. Visualización en el receptor 46

3.3.3. Inserción de datos en el flujo de transporte 53

3.3.3.1. Data piping 55

3.3.3.2. Data streaming 56

3.3.3.3. Data carousel 57

3.3.3.4. Object carousel 57

3.3.3.5. Multiprotocol encapsulation (MPE) ... 57
3.3.4. Transporte de paquetes IP dentro del TS MPEG-2 ... 61
3.4. DVB-RCS (*DIGITAL VIDEO BROADCASTING – RETURN CHANNEL SATELLITE*) ... **63**
3.4.1. Modelos de referencia para redes interactivas por satélite DVB ... 65
3.4.1.1. Modelo de pila de protocolos ... 65
3.4.1.2. Modelo del sistema ... 66
3.4.1.3. Modelo de referencia para redes interactivas por satélite ... 67
3.4.2. Enlace directo (Forward link) ... 69
3.4.3. Canal de retorno (Return link) ... 69
3.4.3.1. Esquema de transmisión de un RCST ... 69
3.4.3.2. Sincronización del RCST ... 70
3.4.3.3. Formato de ráfaga ... 71
3.4.3.4. Aleatorización para dispersar la energía ... 74
3.4.3.5. Codificación ... 75
3.4.3.6. Modulación ... 75
3.4.3.7. Mensajes MAC ... 76
3.4.4. Acceso al medio de la red DVB-RCS ... 77
4. SEGURIDAD EN LA RED VSAT DVB-S ... **79**
4.1. SEGURIDAD *COMMON SCRAMBLING ALGORITHM* ... **80**
4.2. SEGURIDAD *INDIVIDUAL SCRAMBLING* ... **82**
4.3. SEGURIDAD A NIVEL DE RED ... **83**
4.4. SEGURIDAD EN LAS CAPAS SUPERIORES ... **87**
4.4.1. Security Sockets Layer / Transport Security Layer ... 88
4.4.1.1. Estructura protocolo SSL/TLS ... 89
4.4.2. Secure SHel ... 92
4.4.2.1. Estructura del protocolo SSH ... 94
4.4.3. Algoritmos de cifrado ... 99
5. SIMULACIÓN DE UNA RED VSAT CON DVB-S/RCS ... **101**
5.1. "SIMULADOR DE DVB-S" ... **102**
5.1.2. Manual de usuario del "Simulador de DVB-S" ... 103
6. TRANSFERENCIA DE FICHEROS EN UNA RED DVB-S/RCS ... **115**
6.1. *FILE TRANSPORT PROTOCOL* ... **115**
6.2. MODO DE CONEXIÓN ... **116**
6.2.1. Modo activo ... 116
6.2.2. Modo pasivo ... 117
6.3. MODOS DE ACCESO AL SERVIDOR ... **118**
6.3.1. Acceso anónimo ... 119

6.3.2. Acceso de usuario 119
6.3.3. Acceso invitado 119
6.4. TIPO DE TRANSFERENCIA DE DATOS 119
6.4.1. Tipo ASCII 119
6.4.2. Tipo Binario 120
6.5. SEGURIDAD FTP 120
6.5.1. File Transfer Protocol over SSL/TLS 120
6.5.2. SSH File Transfer Protocol 122
7. DISEÑO DE UNA APLICACIÓN SFTP OPERATIVA EN UNA RED DVB-S/RCS 125
7.1. LENGUAJE DE LA APLICACIÓN C# 126
7.1.1. Clase SFTP en C# 127
7.2. ESTRUCTURA DE LA APLICACIÓN 129
7.3. DIAGRAMAS DE LA APLICACIÓN 136
7.3.1. Diagrama conexión estación 137
7.3.2. Diagrama de comprobación de ficheros 137
7.3.3. Diagrama de descarga automática 138
7.3.4. Diagrama de edición o adición de una estación 139
7.3.5. Diagrama de ver datos de una estación 140
7.3.6. Diagrama de búsqueda de ficheros 141
8. PRUEBA DE CAMPO: TRANSFERENCIA DE FICHEROS POR EL PROTOCOLO SFTP EN UNA RED VSAT CON EL ESTANDAR DVB-S/RCS 143
8.1. CONFIGURACIÓN DE LOS EQUIPOS 144
8.2. PRUEBA DE CONEXIÓN 145
8.3. PRUEBA DE TRANSFERENCIA DE DATOS SFTP 147
8.3.1. Configuración del servidor SFTP con Core FTP 147
8.3.2. Configuración del cliente SFTP 150
8.3.3. Transferencia de ficheros en la red SAICA 152
8.3.4. Análisis de paquetes con WireShark 154
9. CONCLUSIONES 159
10. FUTURAS LÍNEAS DE TRABAJO 161
11. BIBLIOGRAFÍA 163
MANUAL SERVIDOR SFTP CORE FTP 166
CREACIÓN DE UN SERVIDOR SFTP 168
CREACIÓN DE UN USUARIO EN UN SERVIDOR SFTP 172
CÓDIGO APLICACIÓN CLIENTE SFTP 179

ÍNDICE

CLASE PRINCIPAL.CS **179**
CLASE CONFIGURACION.CS **190**
CLASE ADDESTACION.CS **196**
CLASE DATOS ESTACION.CS **201**
CLASE ACERCADE.CS **204**

Índice de figuras

FIGURA 1. ESTACIONES TRONCALES Y VSAT 4
FIGURA 2. COMUNICACIÓN ENTRE VSAT 5
FIGURA 3. RED EN MALLA 6
FIGURA 4. RED EN ESTRELLA 7
FIGURA 5. RED UNIDIRECCIONAL 9
FIGURA 6. RED UNIDIRECCIONAL CON CANAL DE RETORNO TERRESTRE 9
FIGURA 7. RED BIDIRECCIONAL 10
FIGURA 8. ESTACIÓN VSAT 11
FIGURA 9. COMPONENTES DE UNA ESTACIÓN VSAT 12
FIGURA 10. EQUIPO ODU DE UNA ESTACIÓN VSAT 12
FIGURA 11.EQUIPO IDU DE UNA ESTACIÓN VSAT 14
FIGURA 12. HUB DE UNA RED VSAT 15
FIGURA 13. *TCP-SPLITTING* 18
FIGURA 14. *TCP-SPOOFING* 19
FIGURA 15. CLASIFICACIÓN DE LOS ESTÁNDARES DE TRANSMISIÓN 21
FIGURA 16. ORGANIGRAMA DEL DVB 26
FIGURA 17. FORMACIÓN DE LOS FLUJOS DE SEÑAL MPEG-2 28
FIGURA 18. UNIDADES BÁSICAS DE MPEG 30
FIGURA 19. CONVERSIÓN DE UN ES EN UN PES 30
FIGURA 20. SINTAXIS DEL PAQUETE PES 32
FIGURA 21. FORMACIÓN DE LOS PAQUETES DE TRANSPORTE 33
FIGURA 22. CONFORMACIÓN DEL MULTIPLEX DE TRANSPORTE TS 34
FIGURA 23. SINTAXIS DEL PAQUETE DE TRANSPORTE 35
FIGURA 24.ESTÁNDARES DE TV DIGITAL POR SATÉLITE 37
FIGURA 25. DIAGRAMA DE BLOQUES FUNCIONAL DEL SISTEMA DVB-S 38
FIGURA 26. CODIFICACIÓN DE CANAL DE DVB-S 40
FIGURA 27. ALEATORIZADOR DVB-S DEL FLUJO DE TRANSPORTE 41
FIGURA 28. PAQUETE TS PROTEGIDO POR REED-SOLOMON RS (204, 188,8) 42
FIGURA 29. ENTRELAZADO Y DESENTRELAZADO CONVOLUCIONAL EN DVB-S 43
FIGURA 30. ENTRAMADO DVB-S 44
FIGURA 31. CODIFICACIÓN INTERNA Y MODULACIÓN QPSK EN DVB-S 45
FIGURA 32. VISUALIZACIÓN DE LOS PROGRAMAS EN EL RECEPTOR DVB-S 46
FIGURA 33. PAT (*PROGRAM ASSOCIATION TABLE*) 48
FIGURA 34. PMT (*PROGRAM MAP TABLE*) 48
FIGURA 35. CAT (*CONDITIONAL ACCESS TABLE*) 49
FIGURA 36. MÉTODOS PARA TRANSPORTAR DATOS SOBRE EL FLUJO MPEG-2 54

ÍNDICE

FIGURA 37. *DATA PIPING* 55

FIGURA 38. *DATA STREAMING* 57

FIGURA 39. ENCAPSULACIÓN MULTIPROTOCOLO (MPE) 59

FIGURA 40. RELACIÓN ENTRE EL PAQUETE DE TRANSPORTE TS, LA SECCIÓN MPE Y EL DATAGRAMA IP 60

FIGURA 41.SINTAXIS DE UNA SECCIÓN MPE-IP 60

FIGURA 42. CAMBIO DINÁMICO DE DIRECCIONES MAC EN MPE 61

FIGURA 43. CABECERA MÍNIMA ULE 62

FIGURA 44. ENCAPSULADO DE UNA SNDU EN PAQUETES DE TRANSPORTE MPEG-2 UTILIZANDO EL PROCEDIMIENTO DE RELLENO 63

FIGURA 45. ENCAPSULADO DE UNA SNDU EN PAQUETES DE TRANSPORTE MPEG-2 UTILIZANDO EL PROCEDIMIENTO DE EMPAQUETAMIENTO 63

FIGURA 46. ESQUEMA DE ENLACE DVB-S/RCS 64

FIGURA 47. ESTRUCTURA EN CAPAS DEL MODELO SIMPLIFICADO DE REFERENCIA PARA DVB-S/RCS 66

FIGURA 48. MODELO DE REFERENCIA DE UN SISTEMA GENÉRICO PARA SISTEMAS INTERACTIVOS 66

FIGURA 49. MODELO DE REFERENCIA PARA REDES INTERACTIVAS POR SATÉLITE 68

FIGURA 50. DIAGRAMA DE BLOQUES DEL PROCESAMIENTO DE LA SEÑAL EN BANDA BASE DEL CANAL DE RETORNO DEL RCST 70

FIGURA 51. COMPOSICIÓN DE UNA RÁFAGA DE TRÁFICO ATM 72

FIGURA 52. COMPOSICIÓN DE UNA RÁFAGA DE TRÁFICO MPEG2 72

FIGURA 53. COMPOSICIÓN DE UNA RÁFAGA SYNC 73

FIGURA 54. COMPOSICIÓN DE UNA RÁFAGA ACQ 74

FIGURA 55. COMPOSICIÓN DE UNA RÁFAGA CSC 74

FIGURA 56. PROCESADO DE LA SEÑAL DESPUÉS DEL CODIFICADOR EN DVB-RCS 75

FIGURA 57. DIAGRAMA DE CONSTELACIÓN PARA QPSK CON CÓDIGO DE GRAY 76

FIGURA 58. MF-TDMA CON SLOTS FIJOS 77

FIGURA 59. MF-TDMA CON SLOTS DINÁMICOS 78

FIGURA 60. CAPAS DE SEGURIDAD EN UNA RED INTERACTIVA DVB-S/RCS 79

FIGURA 61. BLOQUE DE CIFRADO CSA 80

FIGURA 62. ESQUEMA *DESCRAMBLING* DVB-S. 81

FIGURA 63. ESTABLECIMIENTO DE LA SEGURIDAD EN DVB-RCS 83

FIGURA 64. ESTRUCTURA PROTOCOLO AH 85

FIGURA 65. ESTRUCTURA PROTOCOLO ESP 86

FIGURA 66. MODO TRANSPORTE 87

FIGURA 67. MODO TÚNEL 87

FIGURA 68. INICIALIZACIÓN PROTOCOLO SSL/TLS 89

FIGURA 69. ESTRUCTURA PROTOCOLO SSL/TLS 90

FIGURA 70. ENCAPSULADO PROTOCOLO SSL/TLS 92

FIGURA 71. ESTRUCTURA PROTOCOLO SSH 94

FIGURA 72. ESTRUCTURA PAQUETE SSH 96

FIGURA 73. PROTOCOLO DE CONEXIÓN SSH 98
FIGURA 74. MAPA RED SAICA 102
FIGURA 75. VENTANA PRINCIPAL SIMULADOR DE DVB-S 103
FIGURA 76. ESTACIÓN REMOTA DE TALAVERA DE LA REINA (TOLEDO) 104
FIGURA 77. ESTACIÓN REMOTA EN ZORITA (GUADALAJARA) 104
FIGURA 78. PARÁMETROS DE LA CALIDAD DEL AGUA 105
FIGURA 79. ANALIZADOR MULTIPARAMÉTRICO 105
FIGURA 80. ESTACIÓN CENTRAL DE MADRID 106
FIGURA 81. DATOS ESTACIÓN CENTRAL 106
FIGURA 82. PANEL DE CONTROL 107
FIGURA 83. OPCIÓN GUARDAR DEL MENÚ SUPERIOR 108
FIGURA 84. OPCIÓN CARGAR DEL MENÚ SUPERIOR 108
FIGURA 85. BARRA DE HERRAMIENTAS 108
FIGURA 86. DATOS DE UNA TRAMA SAICA 109
FIGURA 87. PAQUETE TCP 109
FIGURA 88. PILA DE DATOS 110
FIGURA 89. PAQUETE TCP 110
FIGURA 90. BOTÓN DE INICIO 110
FIGURA 91. PAQUETE IP 111
FIGURA 92. SECCIÓN MPE 111
FIGURA 93. PAQUETE DE TRANSPORTE DE *TRANSPORT STREAM* 112
FIGURA 94. VENTANA DE CONFIGURACIÓN DE EQUIPOS 114
FIGURA 95. MODO ACTIVO FTP 117
FIGURA 96. MODO PASIVO FTP 118
FIGURA 97. ESTRUCTURA DE UN PAQUETE SFTP 122
FIGURA 98. RED VSAT SAICA 125
FIGURA 99. VENTANA PRINCIPAL DE LA APLICACIÓN SFTP 129
FIGURA 100. OPCIONES DEL MENÚ SUPERIOR 130
FIGURA 101. OPCIONES DE TRANSFERENCIA DE DATOS 130
FIGURA 102. VENTANA DE ESTACIONES 130
FIGURA 103. BÚSQUEDA DE FICHEROS 131
FIGURA 104. VENTANA DEL LOG 131
FIGURA 105. VENTANA DE CONFIGURACIÓN 132
FIGURA 106. VENTANA DE NUEVA ESTACIÓN 133
FIGURA 107. VENTANA DE EDITAR ESTACIÓN 134
FIGURA 108. VENTANA DE DATOS DE UNA ESTACIÓN 135
FIGURA 109. DIAGRAMA DE CONEXIÓN CON UNA ESTACIÓN 137
FIGURA 110. DIAGRAMA DE COMPROBACIÓN DE FICHEROS 138

FIGURA 111. DIAGRAMA DE DESCARGA AUTOMÁTICA 139
FIGURA 112. DIAGRAMA DE EDICIÓN O ADICIÓN DE UNA ESTACIÓN 140
FIGURA 113. DIAGRAMA PARA VER LOS DATOS DE UNA ESTACIÓN 141
FIGURA 114. DIAGRAMA DE BÚSQUEDA DE FICHEROS 142
FIGURA 115. ESQUEMA GRÁFICO DE LA PRUEBA REALIZADA EN LA RED SAICA 143
FIGURA 116. CONFIGURACIÓN IP DE LA ESTACIÓN REMOTA (SERVIDOR SFTP) 144
FIGURA 117. CONFIGURACIÓN IP DE LA ESTACIÓN CENTRAL (CLIENTE SFTP) 145
FIGURA 118. PING DESDE LA ESTACIÓN REMOTA A LA ESTACIÓN CENTRAL 146
FIGURA 119. PING DESDE LA ESTACIÓN CENTRAL A LA ESTACIÓN REMOTA 146
FIGURA 120. VENTANA DE CREACIÓN DE UN SERVIDOR SFTP 148
FIGURA 121. VENTANA CON LA CREACIÓN DE CLAVES PRIVADAS 150
FIGURA 122. VENTANA DE CONFIGURACIÓN DEL CLIENTE SFTP 151
FIGURA 123. VENTANA CREACIÓN DE UNA ESTACIÓN EN EL CLIENTE SFTP 152
FIGURA 124. VENTANA PRINCIPAL DE LA TRANSFERENCIA DE FICHEROS. 153
FIGURA 125. VENTANA EMERGENTE DE FINALIZACIÓN DE TRANSFERENCIA DE DATOS CON LAS ESTACIONES REMOTAS 153
FIGURA 126. PAQUETES CAPTURADOS CON WIRESHARK. 154
FIGURA 127. IDENTIFICACIÓN DE LOS PROTOCOLOS SSH 155
FIGURA 128. INTERCAMBIO DE CLAVES *DIFFIE-HELLMAN*. 155
FIGURA 129. TRANSFERENCIA DE PAQUETES CON ENCRIPTACIÓN SSH. 157
FIGURA 130. ESTRUCTURA PAQUETE SSH ENCRIPTADO. 157
FIGURA 131. PAQUETE TCP CON LA ACCIÓN *RESET*. 158
FIGURA 132. VENTANA PRINCIPAL DEL SERVIDOR CORE FTP 167
FIGURA 133. VENTANA *SETUP* DEL SERVIDOR CORE FTP. 168
FIGURA 134. VENTANA PARA LA CREACIÓN DE UN SERVIDOR. 169
FIGURA 135. VENTANA PARA *SETUP* PARA CREAR UN USUARIO. 173
FIGURA 136. VENTANA PARA LA CREACIÓN DE UN USUARIO EN UN SERVIDOR. 174
FIGURA 137. VENTANA DE PERMISOS DE UN USUARIO. 175
FIGURA 138. VENTANA *SCRIPS/COMMANDS* DE UN USUARIO. 177
FIGURA 139. VENTANA DE SEGURIDAD DE UN USUARIO. 178

Índice de tablas

TABLA 1. TABLAS DEL SI (*SERVICE INFORMATION*) 50
TABLA 2. TIPOS DE MENSAJES SSL/TLS 91
TABLA 3. TIPO DE MENSAJES PROTOCOLO DE TRANSPORTE SSH 97
TABLA 4. TIPO DE MENSAJES PROTOCOLO DE AUTENTICACIÓN SSH 98
TABLA 5. TIPOS DE MENSAJES PROTOCOLO DE CONEXIÓN SSH 99

1. INTRODUCCIÓN

La tecnología ha evolucionado a lo largo de la historia para intentar cubrir las diferentes necesidades que se les presentaban a las personas en cada momento. Un ejemplo de esto es la evolución de las redes de comunicaciones.

La necesidad de los usuarios de interactuar y no ser un mero espectador en las comunicaciones, ha hecho que redes que inicialmente fueron concebidas para la difusión tengan que ser adaptadas para tener una cierta interactividad. Éste es el caso de las redes VSAT basadas en el sistema DVB-S.

Tradicionalmente, para que una red satelital como la red VSAT DVB-S fuera interactiva, había que hacer uso de un canal de retorno provisto por un operador terrestre, que generalmente era distinto al proveedor satelital. Este esquema limitaba mucho las posibilidades de este tipo de redes ya que la principal ventaja de una red satelital es su facilidad para poder ser instalada en cualquier lugar, independientemente de las infraestructuras terrestres construidas. Al depender de un canal de retorno terrestre, tiene que haber una infraestructura que lo soporte.

Para salvar este problema y aprovechar todo el potencial de las comunicaciones por satélite, surgió el estándar DVB-RCS, el cual provee un canal de retorno satelital que evita tener dependencia con la infraestructura terrestre, permitiendo tener redes totalmente interactivas por satélite. El camino de retorno terrestre pasaba a un segundo lugar, empleándose como medio alternativo por si la conexión satelital fallaba o para situaciones específicas. Ahora la infraestructura terrestre no es una condición necesaria.

El objetivo del presente proyecto es hacer un estudio del comportamiento de las redes VSAT que trabajan con DVB-S/RCS. Este tipo de redes ha tenido un gran auge en los últimos tiempos gracias a que al emplear un estándar abierto como es el DVB-S/RCS permite trabajar con gran operatividad y bajo coste.

Para comprender este sistema de comunicaciones, el proyecto está estructurado en dos partes.

En la primera parte se hace una revisión de cómo han evolucionado las comunicaciones satelitales VSAT, indicando las ventajas y desventajas de su implementación y sobre todo la orientación que éstas muestran a la utilización de los estándares DVB. Posteriormente se realiza un estudio de los estándares DVB-S y DVB-RCS en donde se profundiza en conceptos claves tales como el Multiplexado de Transporte MPEG-2, los mecanismos de envío de mensajes de señalización, etc. Todo esto queda explicado con el desarrollo de una aplicación que muestra de manera visual como funciona este tipo de redes bajo una arquitectura de cliente-servidor.

En la segunda parte del proyecto se presta atención a la seguridad de la red, analizando los mecanismos propios que presenta el estándar DVB así como los diferentes protocolos de seguridad existentes en las capas superiores para una protección adicional.

Para comprobar todo el proyecto se realiza una aplicación SFTP en el lenguaje C# con las técnicas de seguridad más fiables. Finalmente se realizan pruebas de conexión en la Confederación Hidrográfica del Tajo y se prueba la aplicación SFTP desarrollada en una red VSAT con DVB-S.

2. COMUNICACIONES POR SATÉLITE BASADAS EN LAS REDES VSAT

Tradicionalmente los satélites de comunicaciones se han usado para establecer enlaces troncales que transporten circuitos telefónicos conmutados, circuitos alquilados y canales de televisión punto a punto. Más recientemente, los avances tecnológicos y el uso de frecuencias más altas han permitido reducir el tamaño y el coste de los terminales, haciendo posible el acceso directo de los usuarios al satélite. Aquí es donde surgen las estaciones VSAT (*Very Small Aperture Terminal*), que son un paso intermediario en busca de esa reducción de tamaño.

Las estaciones VSAT están en el nivel más bajo de una línea jerárquica que ofrece una gran variedad de servicios de comunicaciones. En el nivel más alto se encuentran las conocidas como Estaciones Troncales, cuyo objetivo es el de establecer enlaces de gran capacidad entre sí. Dichas estaciones son usadas principalmente por redes conmutadas internacionales para soportar servicios telefónicos entre países, posiblemente ubicados en diferentes continentes. Son muy costosas y requieren obras de gran envergadura para su instalación, por lo que están administradas por operadores nacionales o por grandes compañías privadas de comunicación.

Las estaciones VSAT son estaciones pequeñas con antenas de entre 2 y 3 metros de diámetro. El pequeño tamaño de las antenas limita la potencia a transmitir y la posibilidad de crear enlaces de gran capacidad, pero gracias a su bajo precio de fabricación y a su fácil instalación convierte a las VSAT en una opción muy atractiva para la comunicación vía satélite. Con las estaciones VSAT se evita tener que usar una red pública para acceder a una estación remota. A este procedimiento se le conozca con el nombre de *bypass*. Toda esta línea jerárquica queda representada en la figura 1.

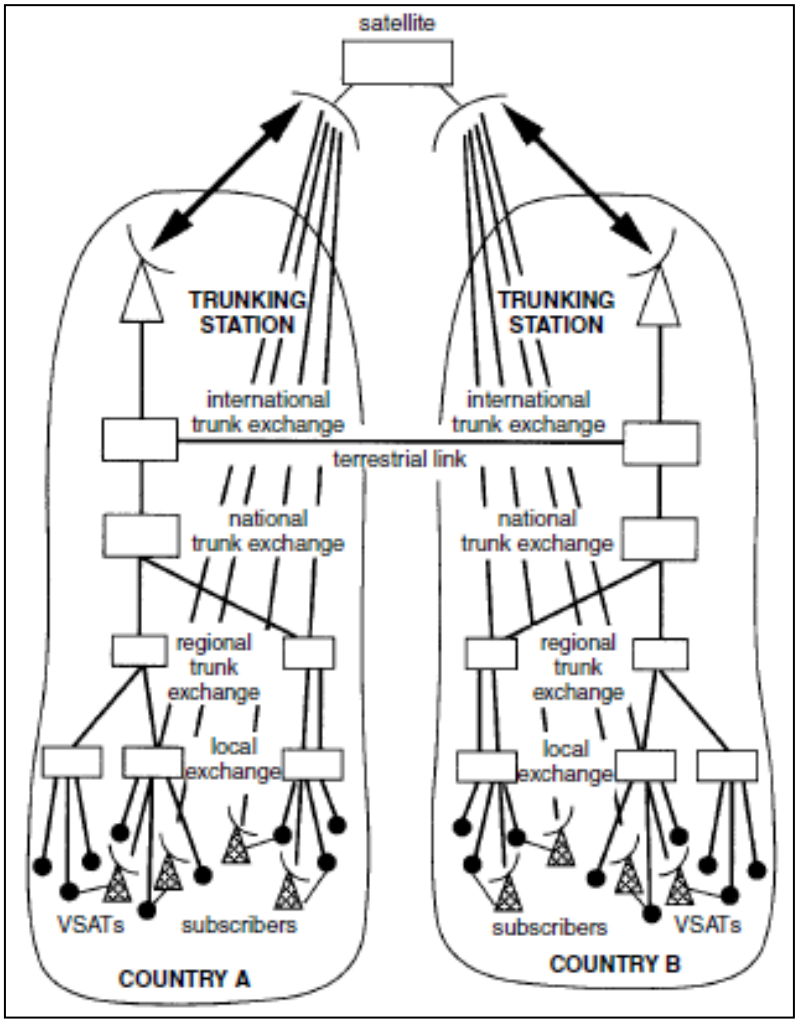


Figura 1. Estaciones troncales y VSAT

En general, las redes por satélite se caracterizan por:

- **Cobertura extensa**, con **rapidez de instalación** de los terminales y **coste independiente** de la distancia dentro de la zona cubierta. Un mismo satélite puede incluir varios haz con diferentes coberturas.
- **Gran capacidad** en el sentido de bajada hacia los terminales (*downstream*) y generalmente menor en el de subida hacia el satélite (*upstream*). La capacidad depende de las características de cada sistema y en particular del tamaño de las antenas.
- **Adecuación** para los servicios de difusión.

2.1. Configuración de las redes VSAT

Las estaciones VSAT se conectan mediante enlaces de radiofrecuencia vía satélite, figura 2. El enlace que une la estación transmisora con el satélite se conoce como *uplink* y al que va desde el satélite a la estación remota como *downlink*. El enlace global que une una estación con la otra, llamado *hop* (salto), consiste de un *uplink* y un *downlink*.

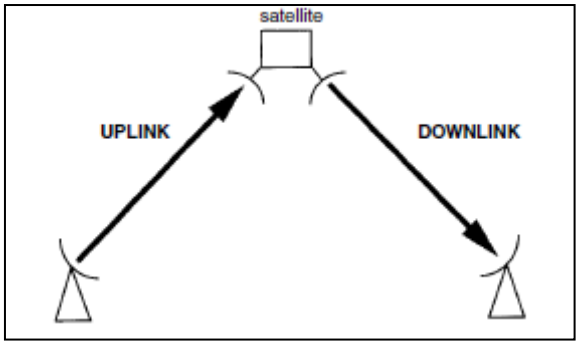


Figura 2. Comunicación entre VSAT

El satélite recibe las señales portadoras que llevan la información modulada de la estación transmisora a través del *uplink*. Una vez en el satélite, dependiendo del tipo de transpondedor (transparente o regenerativo), las señales son amplificadas, cambiadas de frecuencia de portadora a una menor para evitar interferencias y transmitidas de nuevo a la tierra, hacia las estaciones localizadas dentro de su haz de cobertura.

2.1.1. Red en malla

La topología de red mallada, figura 3, es una topología de red en la que cada estación VSAT está conectada directamente con el resto de estaciones que pertenecen a la red. Eso es posible gracias a que todas las estaciones son visibles desde el satélite.

Aunque la comunicación se haga directamente de estación a estación debe de existir un hub que controle la red, configurando y asignando procesos, pero que no está involucrado en el transporte del tráfico. Algunas veces, un terminal VSAT está

equipado con las herramientas de control y gestión, en este caso se dice que la red funciona en *Hublessly*.

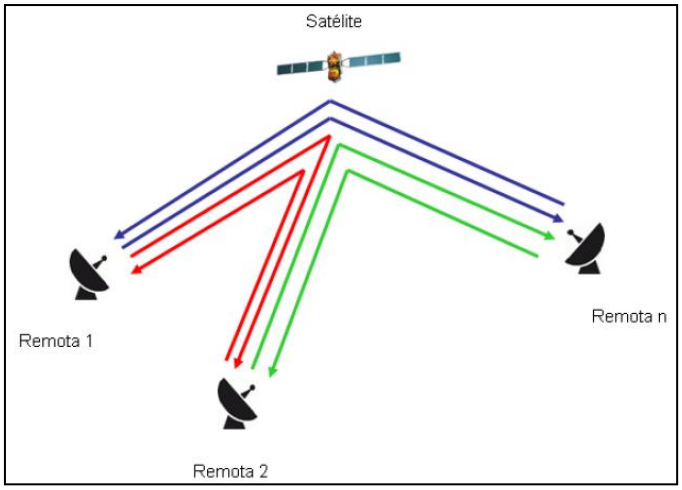


Figura 3. Red en malla

Al configurar una red con esta topología hay que tener en cuenta una serie de limitaciones:

- La gran atenuación que sufre la señal debido a la distancia.
- La potencia limitada de los transpondedores del satélite.
- El tamaño pequeño de las antenas de las estaciones VSAT limita tanto la potencia en el transmisor como la sensibilidad en el receptor.
- Un número elevado de estaciones provocaría mucho tráfico en la red, con el consiguiente aumento de la carga que tendrían que soportar las estaciones.

Estas limitaciones hacen que la señal demodulada en el receptor pueda no cumplir con los requisitos de calidad necesarios para algunas aplicaciones. Es por este motivo por el que los enlaces directos entre estaciones VSAT no se suelen emplear.

La solución a estas limitaciones consiste en la instalación de una estación más grande que una VSAT, llamada hub, que distribuya todo el tráfico por la red

2.1.2. Red en estrella

La topología de red en estrella, figura 4, es una topología en la cual las estaciones están conectadas directamente a un punto central, hub, y todas las comunicaciones se han de hacer necesariamente a través de este.

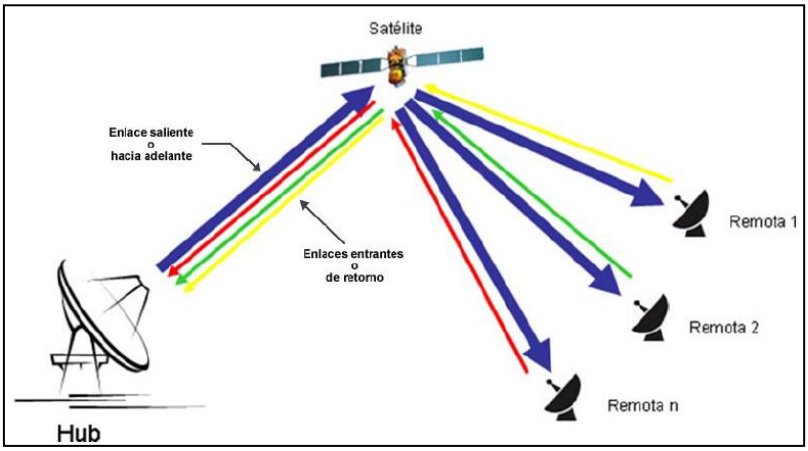


Figura 4. Red en estrella

El hub está pensado para cubrir las carencias de las VSAT. Cuenta con una antena más grande, normalmente entre 3 y 11 metros, resultando esto en una mayor ganancia. Además cuenta con un transmisor más potente.

Como resultado de una capacidad mejorada, el hub es capaz de recibir todas las portadoras provenientes de las estaciones VSAT, extraer la información de las mismas y retransmitirlas hacia las estaciones de destino a través de sus propias portadoras.

Los enlaces que van desde el hub hacia la estación VSAT son llamados *Outbound Links* (enlaces salientes), y desde la VSAT hacia el hub, *Inbound Links* (enlaces entrantes). Cada uno de ellos consiste a su vez, de dos enlaces, un *uplink* y un *downlink*, hacia el satélite y desde él, respectivamente.

La desventaja del empleo de este tipo de redes es debida al **doble salto** que tiene que sufrir la señal para llegar a su destino. Este ***doble salto*** implica un aumento en la latencia, factor crítico en el transporte de aplicaciones sensibles al retardo, tales como los servicios de voz y datos SNA (*Systems Network Architecture*), afectando directamente a la calidad del servicio.

2.1.3. Red hibrida

La topología de red hibrida permite a un grupo de estaciones de una red comunicarse en topología de malla mientras que otras estaciones lo hacen en estrella. Esta topología es empleada en redes donde ciertas estaciones tienen una mayor carga de tráfico entre ellas que con el resto. Las estaciones con mayor demanda de tráfico se configuran en malla para reducir el gasto de equipamiento adicional en el hub y de los recursos satelitales necesarios para un **doble salto**. El resto de la red puede comunicarse con cualquiera de estas estaciones o con las otras por medio de una topología en estrella.

2.2. Clasificación de las redes VSAT

Las redes VSAT se pueden clasificar en tres tipos de redes dependiendo de la forma en la que se comuniquen las estaciones. Existen las redes unidireccionales, las redes bidireccionales y las redes corporativas, las dos primeras se implementan en las redes con configuración en estrella y la última en las redes malladas.

2.2.1. Redes unidireccionales (redes de difusión)

La difusión de contenidos representa una de las primeras y más simples aplicaciones para las redes VSAT. La voz, el video y los datos son transmitidos desde una estación central y difundidos por el satélite a una gran cantidad de estaciones receptoras (estaciones VSAT) que se encuentran dentro de la zona de cobertura. Para evitar el acceso a usuarios no autorizados se manda la señal encriptada, dando lugar al *Narrow Casting* (difusión selectiva). La figura 5 representa un esquema típico de este tipo de redes.

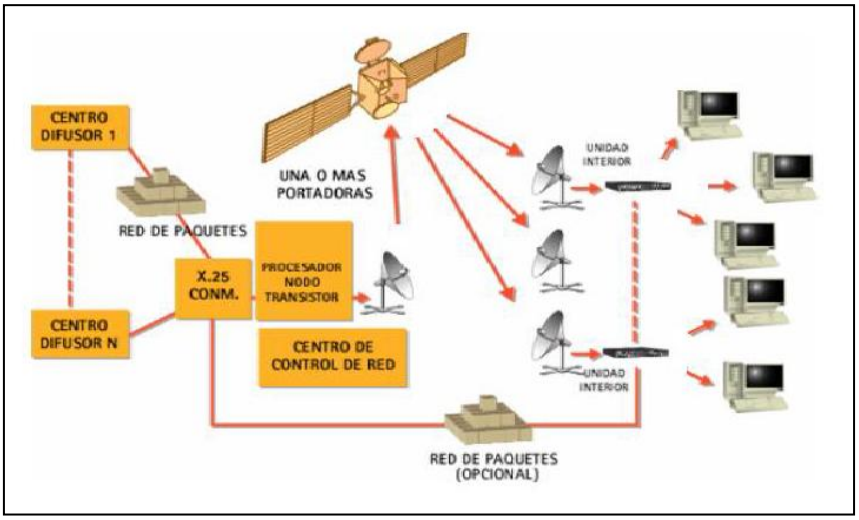


Figura 5. Red unidireccional

A menudo las redes VSAT de difusión emplean un canal de retorno vía terrestre, dando lugar a un sistema hibrido, figura 6. Es muy típico usarlo en servicios como la televisión de pago por satélite (PPV) o Internet para usuarios residenciales.

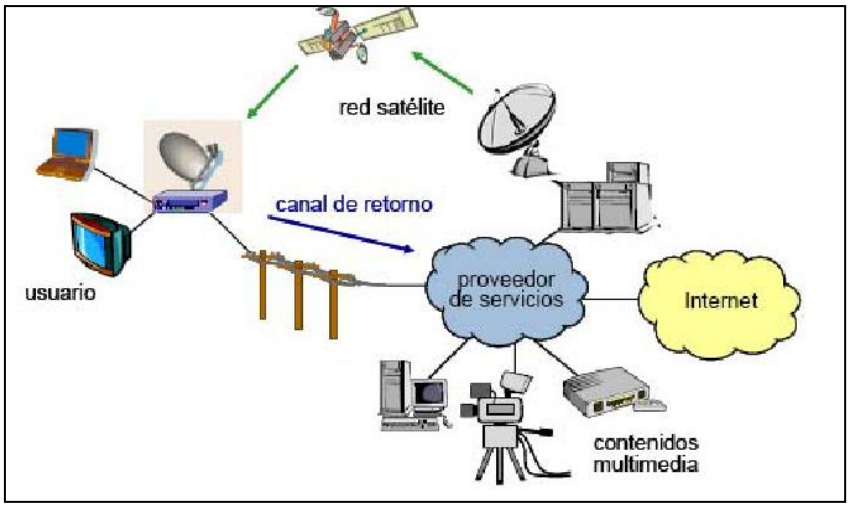


Figura 6. Red unidireccional con canal de retorno terrestre

2.2.2. Redes bidireccionales (redes interactivas)

La arquitectura de estas redes es similar a las unidireccionales, el hub transmite información a las estaciones VSAT a velocidades similares, pero las VSAT también pueden transmitir información empleando el satélite, tal y como se muestra en la figura 7. Ahora el canal de retorno es satelital, siendo normalmente este de una capacidad menor con respecto al de bajada.

Existen dos tipos de transmisión:

- **Hub – Estaciones VSAT**

  La estación central transmite por una o varias portadoras a las estaciones remotas asociadas mediante TDM[1]. La estructura del Multiplex se puede ajustar a la demanda del tráfico, pero siempre reservando una cierta capacidad para los canales de retorno y la asignación del sistema. El número de portadoras de la estación central a las remotas suele ser pequeño y su velocidad de transmisión es proporcionalmente mayor. Por tanto los requisitos de transmisión exigibles a la estación central son mayores.

- **Estaciones VSAT – Hub**

  En este sentido de la transmisión se suele emplear una solución de TDMA[2] por cada portadora. Esto significa que esa estación y sólo esa acceda a la portadora durante un cierto intervalo de tiempo predeterminado con respecto a la referencia de la trama. En el caso de que el tráfico generado por los terminales sea bajo se utiliza un acceso aleatorio con control de colisión.

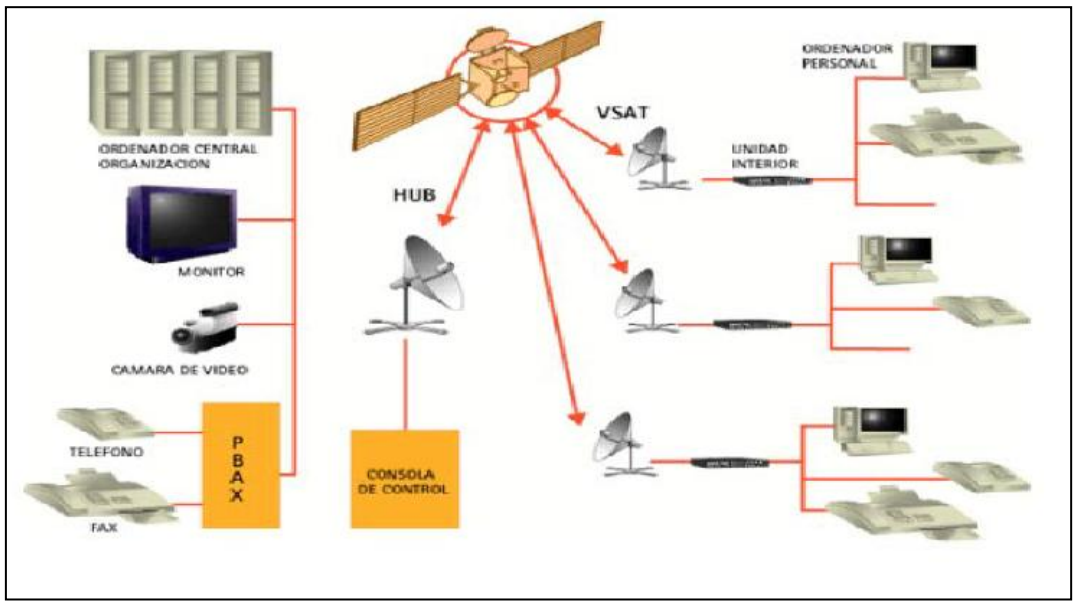


Figura 7. Red bidireccional

Estos sistemas se pueden emplear para crear Redes Privadas Virtuales (PVC) que conecten a varias sucursales de una misma compañía.

[1] **TDM**: Multiplexación por División en el Tiempo.
[2] **TDMA**: Acceso Múltiple por División en el Tiempo.

2.2.3. Redes corporativas

Los sistemas VSAT interactivos limitan las comunicaciones directas entre las estaciones. Cuando se trata de unir varios nodos jerárquicamente iguales y proporcionar servicios digitales avanzados, la utilización del segmento espacial debe ser más eficiente. Para ello se emplean sistemas más potentes que permitan la comunicación directa entre los terminales, empleando una red mallada.

2.3. Estaciones terrestres de las redes VSAT

Hay dos tipos de estaciones terrestres que forman la red VSAT. Por un lado están las estaciones VSAT, que son los terminales que se comunican entre sí, y por otro está el hub, que se encarga de administrar y controlar toda la red.

2.3.1. Estación VSAT

Una estación VSAT, figura 8, es un terminal de dimensiones pequeñas diseñado para la comunicación de datos vía satélite, compuesto por dos equipos: la unidad externa ODU (*Outdoor Unit*) y la unidad interna IDU (*Indoor Unit*).

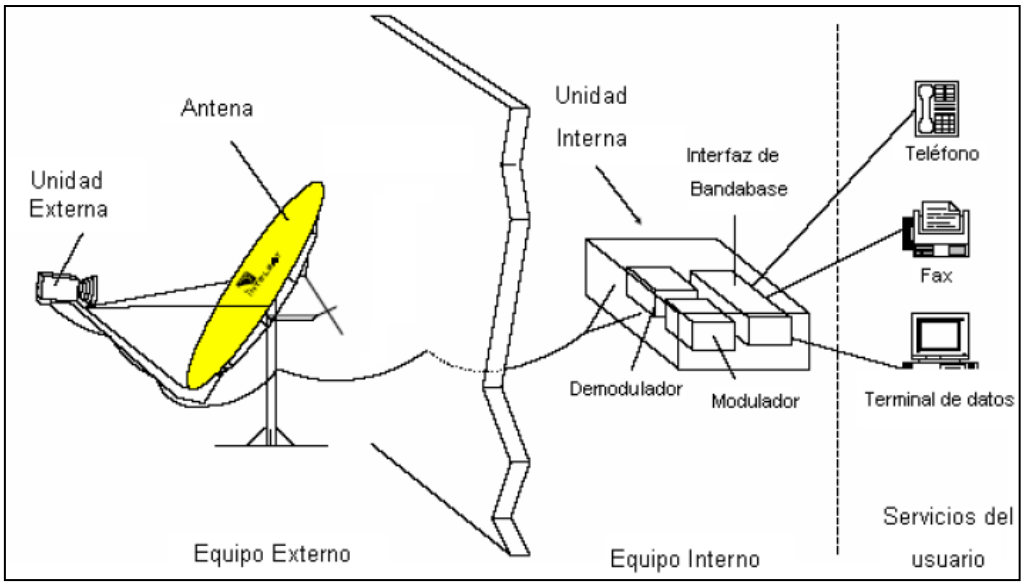


Figura 8. Estación VSAT

La ODU es la interfaz del VSAT con el satélite, mientras que la IDU es el interfaz del VSAT con los terminales del cliente o con una red de área local. En la siguiente imagen se aprecia los diferentes componentes que forman parte de cada unidad.

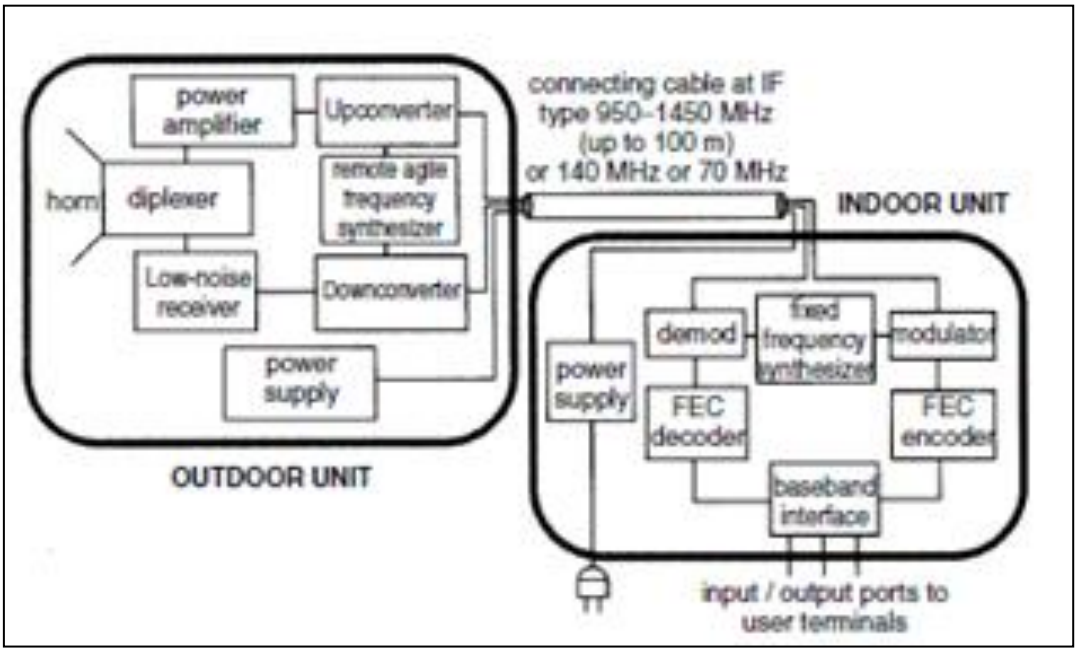


Figura 9. Componentes de una estación VSAT

La unidad externa (ODU)

La ODU, figura 10, es la unidad encargada de la transmisión y recepción de señales hacia o a través del satélite. En la siguiente imagen se muestra la típica unidad exterior utilizada en las comunicaciones VSAT.

Figura 10. Equipo ODU de una estación VSAT

Está compuesta básicamente por los siguientes elementos:

- Antena. La antena es una plato de entre 1.2 y 2.4 metros de diámetro cuya misión es la de concentrar las señales recibidas desde el satélite en un punto (de ahí su forma parabólica) y transmitir las informaciones hacia el mismo.
- Sistemas Electrónicos. Formado por el amplificador de transmisión, el LNB (Amplificador de bajo nivel de ruido), el sintetizador de frecuencia, los osciladores para variar la frecuencia, el duplexor y el amplificador de potencia.

Los parámetros utilizados para evaluar la Unidad Exterior son:

- Las bandas de frecuencia de transmisión y recepción.
- La resolución del transmisor y receptor para ajustar correctamente la frecuencia de la portadora transmitida o para sintonizar la frecuencia de la portadora recibida.
- Nivel de los lóbulos secundarios, que serán importantes para determinar los niveles de interferencia producidos y recibidos.
- La PIRE (Potencia Isotrópica Radiada Equivalente) que condiciona la frecuencia del enlace de subida. Está depende de la ganancia de la antena y de la potencia del transmisor.
- La figura de mérito o factor de calidad (G/T) es un factor que refleja la capacidad para conseguir un alto valor de relación de señal a densidad espectral de potencia, por lo tanto condiciona la frecuencia del enlace de bajada. Esta depende de la ganancia de la antena así como de la temperatura de ruido del sistema.

La unidad interna (IDU)

La IDU es la unidad encargada de la interfaz entre la estación VSAT y el terminal de usuario o LAN, y se encuentra instalada en la localidad del usuario. En la figura 11 se muestra el aspecto típico de una IDU.

Figura 11.Equipo IDU de una estación VSAT

La unidad interior está compuesta básicamente por:

- Sintetizador de frecuencia fija.
- Modulador/Demodulador.
- Puertos de entrada/salida.

Los parámetros utilizados para evaluar la unidad interior son:

- Número de puertos.
- Tipos de los puertos, ya que estos pueden ser mecánicos, eléctricos, funcionales o de procedimiento del interfaz.
- Velocidad de los puertos, la cual representa la máxima velocidad del flujo de datos entre el usuario y la IDU.

Para conectar la unidad exterior con la unidad interior se usa el **cable de enlace de frecuencia intermedia (IFL)**. Estos son cables de tipo coaxial diseñados para ser inmunes a la interferencia y con un ancho de banda suficiente como para operar a frecuencia intermedia.

2.3.2. Estación HUB

El hub es la estación central de una red VSAT y esta no es más que una estación más dentro de la red pero con la particularidad de que es más grande, la antena es del orden de 15 metros de diámetro y maneja mayor potencia de emisión. Debido a su gran coste, la empresa tiene la opción de tenerlo en propiedad o alquilarlo a un operador de servicio.

Existen diferentes tipos de hub:

- **Hub Dedicado**. Un hub dedicado soporta una red única con miles de estaciones VSAT conectadas. Ofrece un control total de la red al cliente, por lo que en periodos de expansión, cambios en la red o problemas resulta la solución más simple. Sin embargo representa la opción más cara y solo es justificado con un número elevado de terminales VSAT.
- **Hub Compartido**. Un hub compartido se basa en que varias compañías alquilan los servicios de un hub a un operador VSAT con el objetivo de compartir gastos. El uso de este tipo de redes tiene una serie de desventajas con respecto al hub dedicado: se necesita conectar el hub con la sede central de la compañía, lo que implica a recurrir a cableado o utilización de la red pública. Además de la limitación a la hora de configurar y ampliar la red, que al no ser propietarios del hub está limitado por el operador VSAT.
- **Mini Hub**. Un mini hub, como su nombre indica, es un pequeño hub con una antena de 2 a 3 metros de diámetro y con un coste mucho menor que un hub dedicado. Un hub de este tipo normalmente soporta alrededor de 300 a 400 estaciones VSAT.

La arquitectura de un hub, figura 12, se divide en dos partes fundamentales: la unidad de radiofrecuencia y la unidad interna. Ambas se explican a continuación:

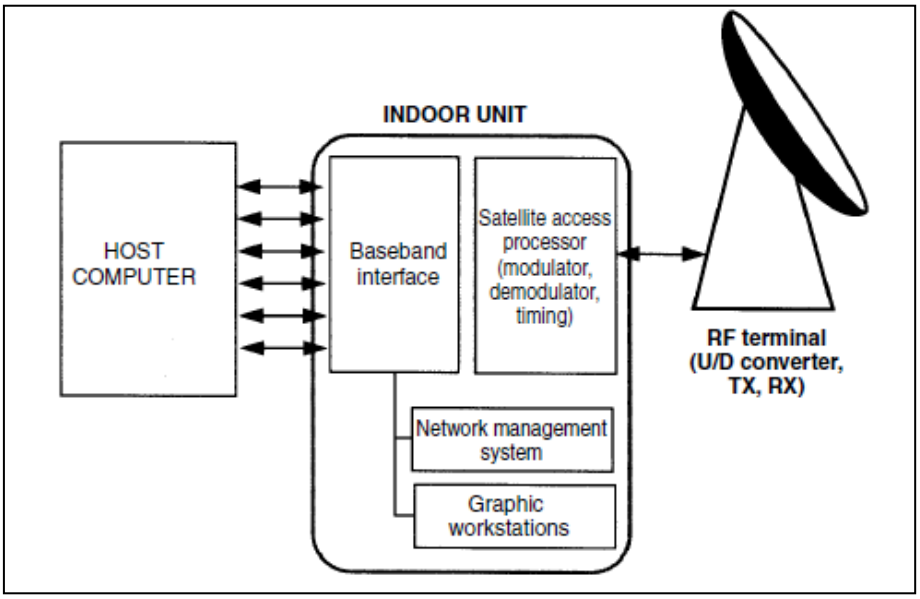


Figura 12. Hub de una red VSAT

a) **Unidad de radiofrecuencia**: al igual que en las estaciones VSAT esta se encarga de la interfaz con el satélite para el envío y recepción de señales.
b) **Unidad interna**: A diferencia de la IDU de la estación VSAT, esta puede actuar como interfaz con otro host o con una red de conmutación, ya sea pública o privada, dependiendo de sí el hub es dedicado o compartido.

La estación hub está provista de un sistema para la administración de la red denominado NMS (*Network Management System*). Este es un ordenador o estación de trabajo equipado con un software especializado, usado para realizar funciones operacionales y administrativas. Está conectado a cada estación VSAT de la red por medio de circuitos virtuales permanentes, por donde se intercambian mensajes de administración constantemente.

Funciones Operativas

Las funciones operacionales están relacionadas con el manejo y configuración dinámica de la red. Se encargan de agregar o eliminar estaciones VSAT, portadoras e interfaces de la red. Aparte incluyen una supervisión y control del funcionamiento del hub, de cada estación VSAT y de todos los puertos de datos asociados a la red. Esto conlleva la gestión de las herramientas que proveen la asignación y conectividad de las VSAT en tiempo real, así como el manejo y control de nuevas instalaciones y configuraciones. Si la capacidad de la red está saturada debido al alto volumen de conexiones que hay, el NMS lo notifica al operador y evita que más usuarios VSAT puedan acceder al servicio.

Funciones Administrativas

La parte administrativa del hub tiene que ver con el inventario del equipo, los archivos de uso de la red, la seguridad y la facturación.

2.4. TCP sobre enlaces satelitales

Los protocolos de transporte usados actualmente en las redes terrestres presentan serios problemas para su implantación en las redes por satélite. Se ha demostrado que TCP tiene una degradación significativa sobre enlaces satelitales debido principalmente a tres factores:

a) **Latencia**: las redes con enlaces satelitales presentan grandes retardos de propagación. Una señal que viaja por medio de un satélite geoestacionario a su destino recorre entre 70000 y 90000 km, dependiendo de la localización geográfica entre el emisor y el receptor. Suponiendo que la señal viaja a la velocidad de la luz, el tiempo en alcanzar al receptor está entre 0.25 y 0,26 sg. Como en toda transmisión de datos fiable, se requiere un reconocimiento de los datos enviados. Esto supone un viaje de vuelta de otros 0,25 sg. Así que el tiempo total de retardo, desde que el paquete es enviado hasta que el transmisor recibe el paquete de reconocimiento está entre 0,5 y 0,6 sg. El transmisor debe retener la información transmitida mientras no reciba un reconocimiento por parte del receptor.
b) **Error**: el uso de enlaces satélites introduce una alta probabilidad de error, aumentando la probabilidad de que los paquetes tengan que ser retransmitidos y con ello el volumen de tráfico.
c) **Asimetría**: gran asimetría entro los canales de difusión y los de retorno.

La degradación que sufre TCP se debe sobre todo al algoritmo de control de congestión, que no está preparado para trabajar con las deficiencias de los enlaces satelitales. TCP estima que si un paquete no ha recibido su confirmación dentro del *timeout* especificado es porque hay congestión en la red y no porque hubiera habido algún error en la transmisión. Entonces, reduce la ventana de congestión a la mitad. Este efecto provoca una reducción innecesaria de la carga del sistema y, por lo tanto, del rendimiento del protocolo TCP. Esta degradación se ve acentuada en sistemas inalámbricos con grandes retrasos.

Se han propuesto algunas soluciones para superar los problemas de TCP sobre enlaces satelitales.

TCP-Splitting

El *TCP-Splitting* es una solución que busca mejorar el rendimiento general de TCP. Se basa en romper una conexión TCP entre dos dispositivos finales mediante la instalación de dos PEPs (*Performance Enhancement Proxies*) en los extremos del segmento espacial. La comunicación generalmente se divide en tres partes: emisor-PEP, PEP-PEP y PEP-receptor, en donde cada segmento es a su vez una conexión TCP. El flujo de datos es enviado a través de los diferentes segmentos hasta llegar al receptor. La figura 13 muestra el esquema que emplea esta técnica.

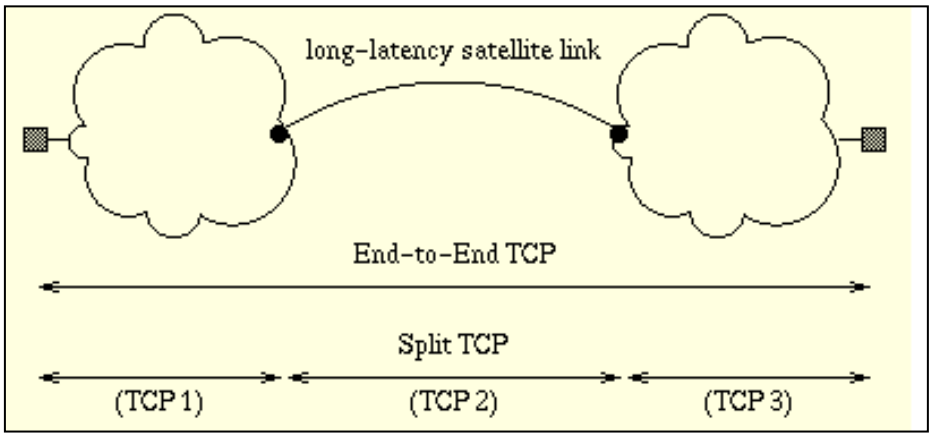


Figura 13. *TCP-Splitting*

TCP-Spoofing

En *TCP-Spoofing* es una solución que se basa en confirmar la recepción del paquete antes de que llegue al receptor. Para ello un dispositivo en el hub reconoce el paquete TCP sin esperar el reconocimiento del paquete actual por el receptor. Esto hace que el transmisor vea una red con baja latencia. El dispositivo intermediario almacena los segmentos reconocidos en el buffer a la espera de que el receptor envié un paquete de reconocimiento. Si lo recibe, el paquete se descarta para evitar duplicaciones, sino, el paquete perdido es retransmitido hacia el receptor. En la figura 14 se muestra como funciona esta solución.

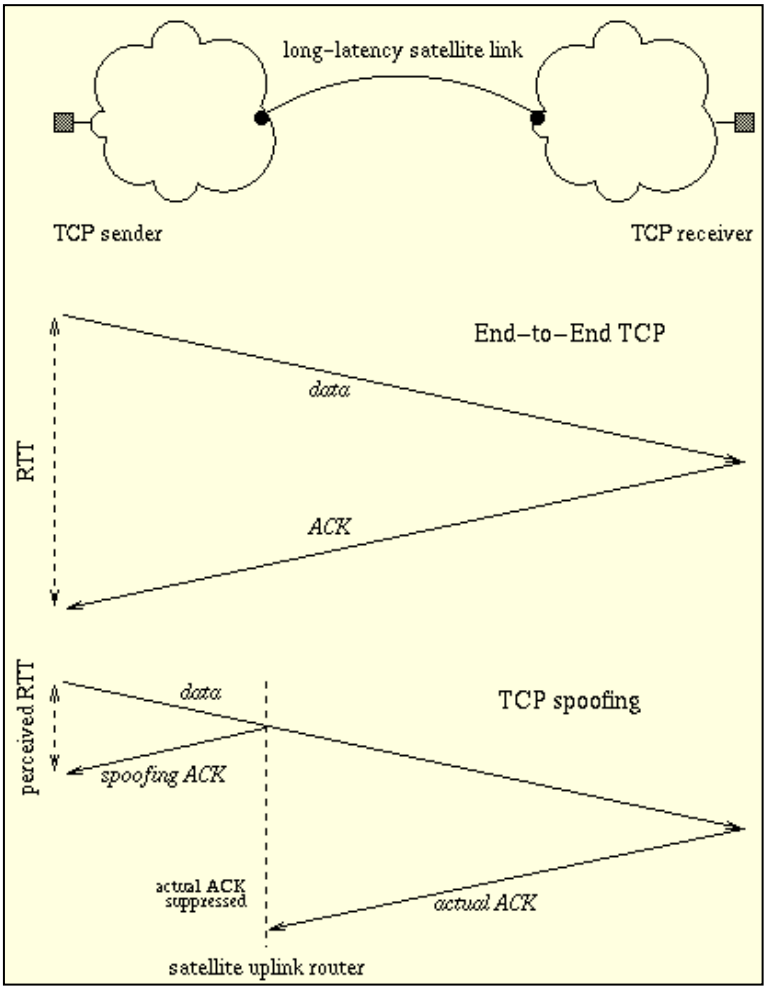


Figura 14. *TCP-Spoofing*

Tanto *TCP-Splitting* como *TCP-Spoofing* eliminan el concepto de protocolo entre dispositivos finales, el emisor puede creer que un segmento ha llegado a su destino cuando en realidad el paquete todavía está en transmisión. Esto es aceptable en muchas aplicaciones, pero puede suponer un problema si una aplicación requiere del concepto de extremo a extremo.

Extensiones del TCP

Las mejoras aplicables al protocolo TCP son varias. Entre ellas cabe destacar el **TCP-LW** (*Large Window*), el **TCP-SACK** (*Selective Acknowledgement*), el **TCP-Peach**, el **TCP-Westwood** y el **TCP-Hybla**.

2.5. Estándares para comunicaciones por satélite

Como en todos los sistemas de comunicaciones, los estándares aseguran que varias partes de un sistema puedan trabajar conjuntamente para proporcionar un servicio.

La mayoría de los satélites de comunicaciones comerciales en la órbita geoestacionaria actúan como simple repetidores, es decir, los satélites solo se encargan de redireccionar el flujo de datos sin tomar parte activa en ellos. Por este motivo, los estándares se aplican del mismo modo que a cualquier red de comunicaciones de datos terrestre.

Nivel Físico

La primera capa dentro de cualquier modelo de red está formada por el nivel físico, en donde las características son independientes del tráfico que se vaya a transportar. Aquí más que en ninguna otra capa es necesaria la compatibilidad ente los equipo para que sea posible la comunicación.

En las comunicaciones digitales por satélite la capa física afecta al equipo de radiofrecuencia, a las antenas y a los enlaces de subida y de bajada. Las características físicas normalmente son específicas para un satélite y son establecidas por el operador del mismo para asegurar que la comunicación sea satisfactoria. Estas normalmente definen:

- Radiofrecuencias y Anchos de Banda
- Potencia Isotrópica Radiada Equivalente (PIRE)
- Nivel de señal a Ruido E_b/N_0
- Ruido y Nivel de los Espurios
- Parámetros de la antena
- Costes del *Uplink* y del *Downlink*

Estos parámetros dictan el diseño de las antenas y el equipo de radiofrecuencia, y son los mismos independientemente de que sistema o servicio utilice el transpondedor. Por lo tanto, diferentes sistemas de comunicaciones pueden emplear el mismo estándar físico ya que utilizan el mismo tipo de satélite.

Nivel de Enlace

La siguiente capa corresponde al nivel de Enlace (Nivel 2 en el modelo OSI, dentro del Nivel 1 en TCP/IP). Es el responsable de la transferencia fiable de información a través de un camino abierto por la capa física.

El objetivo es conseguir que la información fluya libre de errores entre dos máquinas conectadas. Para ello forma paquetes de datos dotados con una dirección de capa de enlace, controla la congestión del medio y vigila que no haya errores en la transmisión.

En las comunicaciones por satélite estas normas son específicas para un sistema y normalmente son establecidas por el operador del sistema para asegurar que las estaciones puedan trabajar juntas. En ellas se indica cómo se llevan a cabo las funciones del DSP (*Digital Signal Proccesing*) y por lo general incluyen:

- Codificación y Decodificación
- Compresión y Descompresión
- Multiplexación y Demultiplexación
- Modulación y Demodulación

Estos parámetros difieren considerablemente dependiendo del sistema o servicio al cual se apliquen. Varios sistemas pueden emplear el mismo estándar físico pero usar diferentes estándares a nivel de enlace y de red.

Los estándares de transmisión de datos se pueden clasificar en tres categorías, tal y como muestra la figura 15:

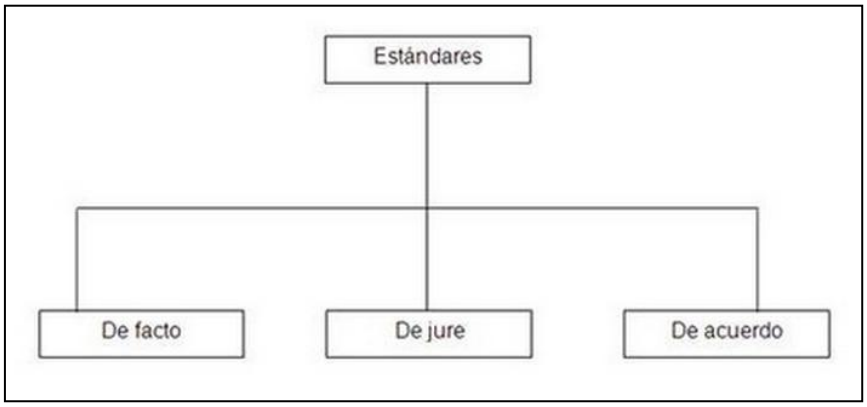


Figura 15. Clasificación de los estándares de transmisión

Estándares De Facto

Son los diseñados por "convención". Estos se pueden dividir en dos clases: propietarios y no propietarios. Los estándares propietarios o cerrados han sido desarrollados por empresas particulares como base del funcionamiento de sus productos. Se les llama propietarios porque solo ellos pueden utilizar el estándar y entorpecen la posibilidad de conectar con otros equipos de distintos proveedores. Un ejemplo de estándar de transmisión por satélite propietario es el *IP Over Satellite* desarrollado por empresa *Hughes Networks Systems.* Los estándares no propietarios o abiertos han sido desarrollados por grupos de trabajos que posteriormente lo han hecho público, facilitando la comunicación entre equipos de diferentes sistemas. Un ejemplo de este tipo de estándar son los desarrollados por el proyecto europeo para la transmisión DVB.

Estándares De Jure

Son los estándares regulados por alguna organización oficial. Los estándares que no han sido aprobados por un organismo reconocido pero que se han adoptado como estándar por su amplio uso son estándares de facto.

Estándares De Acuerdo

Los estándares de facto son definidos por convenio o alianza entre proveedores, fabricantes y usuarios.

En los últimos años está habiendo un gran auge de los estándares abiertos en el mercado común. Para tratar de entender esto se van a exponer las desventajas de los estándares propietarios y las ventajas de los estándares abiertos.

Desventajas de los estándares propietarios

- Una vez instalada una red, solo se puede expandir con equipos del mismo proveedor original, lo que implica que el cliente este atado al fabricante durante toda la vida útil del sistema.

- El comprador debe aceptar las condiciones técnicas, legales y económicas del vendedor.
- No existe interoperabilidad entre las diferentes marcas de equipos.

Ventajas de los estándares abiertos

- Interoperabilidad entre diferentes terminales y equipos, lo que elimina la dependencia del proveedor.
- Reducción de costes, tanto a nivel de terminal como de instalación. Por un lado, al adoptarse universalmente el estándar, se incrementan los volúmenes de producción y consecuentemente se reducen los costes gracias a las economías de escala. Por otro lado, al normalizarse las diferentes arquitecturas de los equipos, las herramientas y los procedimientos, se reducen los costes de instalación y la inversión en preparación del personal técnico es menor.

3. EL PROYECTO DVB

3.1. ¿Qué es el DVB?

El DVB (*Digital Video Broadcasting*) es una organización de origen europeo fundada en 1993, encargada de promocionar estándares de comunicaciones a nivel internacional. Su función es la de regular y diseñar los procedimientos en la transmisión de señales digitales para la difusión de servicios multimedia y de datos.

El *DVB Project* es un consorcio formado por más de 200 organizaciones, instituciones y empresas líderes a nivel mundial (proveedores de servicio, fabricantes, operadores de red, desarrolladores de software, cuerpos reguladores) extendido por más de 35 países cuyo objetivo es diseñar estándares abiertos e interoperables para el desarrollo global de los medios de comunicaciones digitales y de los servicios de radiodifusión.

Está abierto a cualquier organización que esté involucrada en el tema de la radiodifusión digital, y ofrece la posibilidad de participar en el desarrollo de las especificaciones DVB.

Hasta que una norma es considerada como estándar oficial tiene que pasar por una serie de módulos y grupos de trabajo dentro del propio proyecto DVB.

1. El **Modulo Comercial** se encarga de definir y establecer los requisitos comerciales, así como de imponer exigencias funcionales, objetivos de coste y plazos según los requerimientos del mercado.
2. El **Modulo Técnico** elabora unas especificaciones técnica que cumplan con los requisitos establecidos anteriormente.
3. Estas especificaciones son revisadas por el modulo comercial y enviadas a la **Junta Directiva** del DVB para su aprobación final.
4. Una vez recibida la aprobación por el DVB, se envía a la ETSI (*European Telecommunications Standards Institute*) para su publicación como estándar oficial.

Aparte de estos módulos existen otros dos módulos más que forman el conjunto del DVB:

- **Módulo de los derechos de propiedad intelectual**: Encargado de las cuestiones relacionadas con las políticas de protección de los derechos intelectuales del DVB.
- **Módulo de comunicaciones y promociones**: Asegura que todas las partes interesadas estén informadas de las actividades del DVB, es el responsable de las relaciones externas del DVB.

Estos módulos se recogen en la figura 16, que representa como esta estructura el DVB.

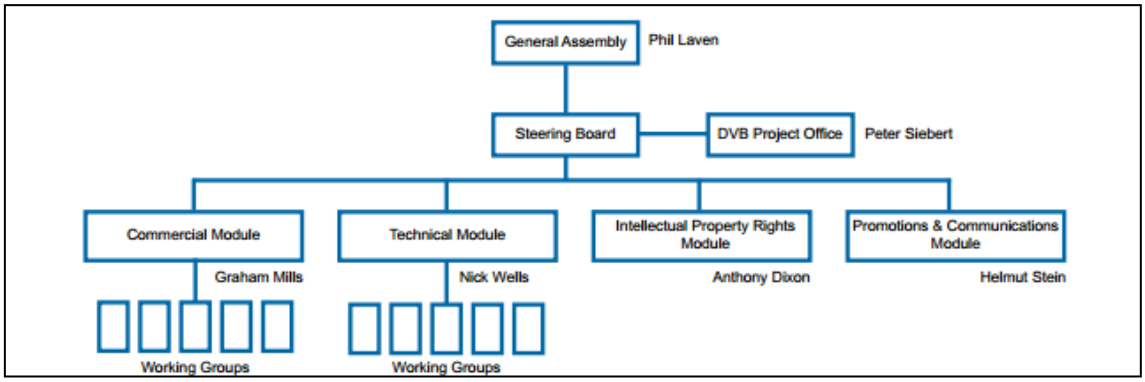


Figura 16. Organigrama del DVB

El DVB ha elaborado distintos estándares en función de las características del medio de difusión.

- Terrestre: DVB-T y DVB-T2, y para dispositivos portátiles el DVB-H
- Cable: DVB-C y DVB-C2
- Satélite: **DVB-S** y DVB-S2, y para terminales portátiles el DVB-SH

Cada estándar define los esquemas de codificación de canal y de modulación, ya que cada medio tiene unas características diferentes, pero todos siguen la codificación fuente del estándar MPEG-2. Más concretamente se define la señal de entrada y salida del sistema, que se denomina flujo de transporte MPEG2 (*MPEG2- Transport Stream*). Dentro de este flujo de transporte se pueden transportar tanto video y audio codificado en MPEG-2, como datos encapsulados.

3.2. El sistema MPEG-2

El MPEG es un grupo de trabajo que se formó con el objetivo de establecer estándares internacionales para la codificación de video y audio en un formato comprimido para su posterior transmisión. Con este fin se ideó un conjunto de estándares recogidos bajo el nombre de MPEG-2 y que han sido publicados como una norma de la ISO en el ISO/IEC 13818-1.

Este sistema es usado como medio de transporte para los sistemas DVB por ser uno de los formatos más sofisticados y aceptados por la industria de las comunicaciones. Así la señal de entrada y de salida especificada para los sistemas DVB es el flujo de transporte MPEG-2 (MPEG-2 TS).

En este apartado se describirán ciertas características del sistema MPEG-2, en especial a las que hacen referencia a la formación del flujo de transporte TS.

3.2.1. Formación de los flujos de señal MPEG-2

Para la obtención de los flujos de señal MPEG-2, los programas tienen que pasar por una serie de bloques. La figura 17 muestra un esquema de dicho proceso.

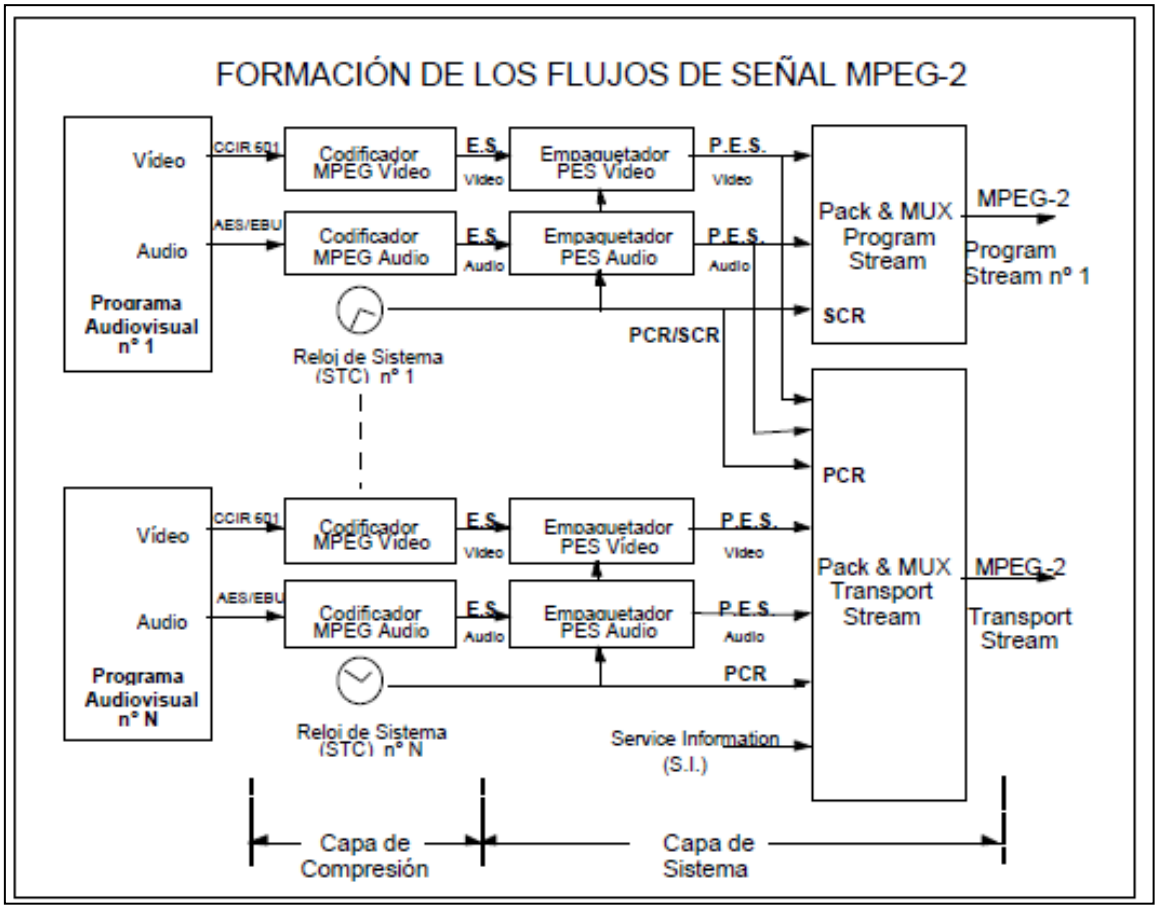


Figura 17. Formación de los flujos de señal MPEG-2

Para entender mejor este esquema conviene dividirlo en dos bloques o capas:

a) **La Capa de Compresión.** Es la encargada de realizar las operaciones de codificación MPEG-2 de los datos que vienen en bruto del dispositivo a través de los procedimientos de compresión audio y video.

b) **La Capa de Sistema**. Es donde se realizan las operaciones necesarias hasta la obtención del flujo de señal MPEG-2. Consiste en la organización en paquetes de los datos comprimidos y la posterior multiplexación de todas las señales pertenecientes al programa en un único flujo de señal.

El componente básico de un flujo MPEG son los flujos elementales ES *(Elementary Streams*) que corresponden a las señales de audio y video una vez comprimidas en la Capa de Compresión. Estos ES pasan entonces a la Capa de Sistema en donde se empaquetan en paquetes de longitud variables conocidos como paquetes PES, formando el flujo elemental paquetizado PES *(Paquetised Elementary Streams)*. Una vez obtenidos los paquetes PES, estos pueden ser multiplexados de dos maneras

diferentes, generando dos flujos de datos diferentes. Por un lado el flujo de programa PS *(Program Stream)* y por otro el flujo de transporte TS *(Transport Stream)*.

- **Flujo de programa PS:** Es el resultado de combinar todos los PES que pertenecen a un único programa y tienen una base común de tiempo. Los paquetes del flujo de programa suelen ser de longitud variable y relativamente grandes. Se utilizan en entornos libres de errores como en los CD-ROM.
- **Flujo de transporte TS:** Es el resultado de combinar los PES de diferentes programas con distintas bases de tiempo en un único flujo. Los paquetes del flujo de transporte son de una longitud fija y relativamente corta (188 bytes) para una buena transmisión en entornos donde se producen errores.

Debido a las limitaciones del flujo de programa de solo poder transmitir un único programa y a su vulnerabilidad en entornos en donde pueden existir errores, los sistemas de difusión DVB utilizan el flujo de transporte TS como medio vehicular.

3.2.2. Formación del flujo de transporte TS

En este apartado se va a detallar el mecanismo por el cual se obtiene el flujo de transporte TS, desde los datos en bruto hasta los paquetes PES que forman el flujo de transporte.

3.2.2.1. Unidades básicas

Como se ha visto en el apartado anterior, el flujo de transporte TS comienza con un programa digitalizado sin comprimir que contiene diferentes tipos de información (Video, Audio…). Cada una de estas informaciones digitales, todavía sin comprimir, se agrupan en bloques elementales denominados Unidades de Presentación *(Presentation Units). E*stas unidades de presentación son codificadas para disminuir la velocidad binaria del flujo de señal. Eso se hace mediante la sustitución de las unidades de presentación por Unidades de Acceso *(Access Units).* La figura 18 muestra las unidades básicas expuestas anteriormente.

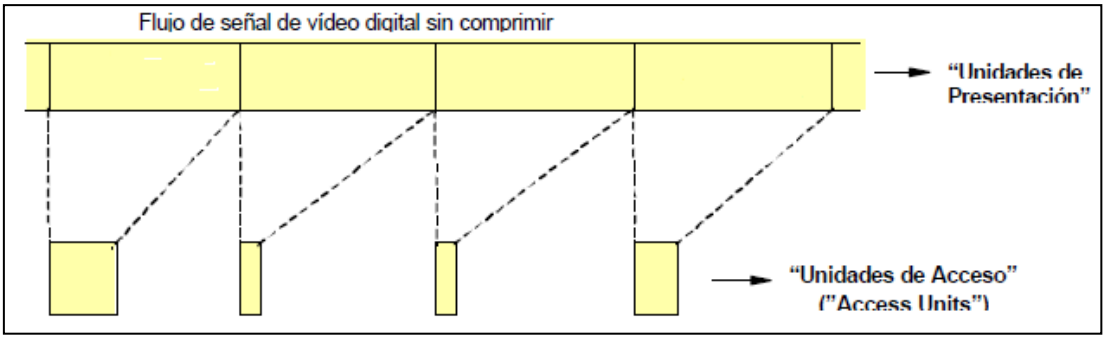


Figura 18. Unidades básicas de MPEG

El resultado de la codificación MPEG de una secuencia de vídeo o audio es una sucesión de unidades de acceso de vídeo o audio respectivamente. Dicha sucesión de unidades de acceso forman lo que se conoce como el "Flujo Elemental de Video" *(Video Elementary Stream)* y el "Flujo Elemental de Audio" *(Audio Elementary Stream)*.

3.2.2.2. Empaquetado PES

En MPEG existen 3 tipos diferentes de flujos elementales o ES:

- **Flujos de video:** Procedentes de la codificación de las señales de video.
- **Flujos de audio:** Procedentes de la codificación de las señales de audio.
- **Flujos de datos:** Contiene cualquier tipo de dato, ya este comprimido o no.

Una vez obtenidos, todos los ES se dividen en paquetes de longitud variable para su transporte. Los flujos resultantes de esta división son los PES, figura 19. Un PES está compuesto íntegramente por paquetes PES *(PES-Packets)*, los cuales pueden tener una longitud máxima de 64KB.

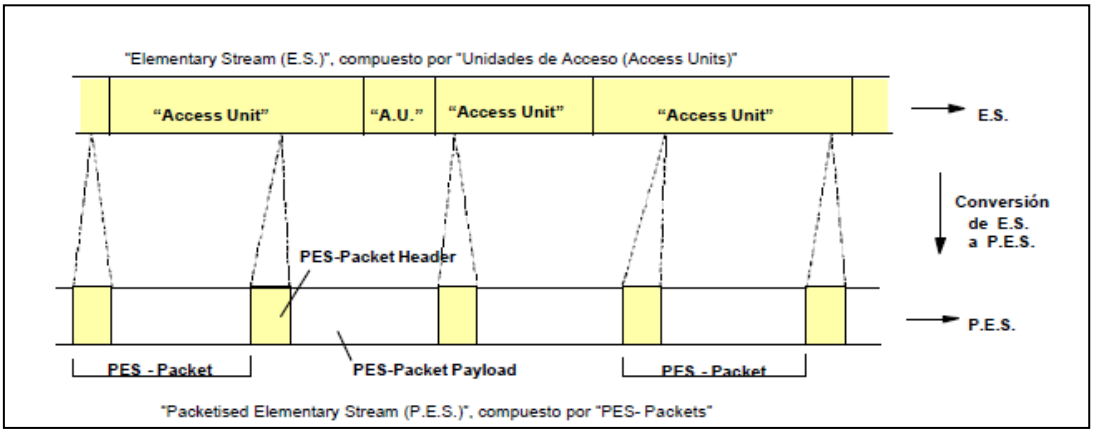


Figura 19. Conversión de un ES en un PES

El paquete PES está formado por una cabecera corta y una carga útil en donde viajan los flujos de audio, video y datos.

Sin embargo, los paquetes PES no son la única manera definida en MPEG-2 para la transmisión de datos, es decir, no todos los flujos elementales son finalmente introducidos en paquetes PES. También se emplean las secciones MPEG-2 para el transporte de ciertos datos.

Estructura del paquete PES (PES-Packet)

Todos los ES son empaquetados en paquetes de longitud variable llamado paquetes PES, figura 20. Estos paquetes, que normalmente son de 64KB, tienen como mínimo un **encabezado** de 6 bytes.

- *Prefijo de Código de Inicio (3 bytes).* Siempre vale '00 00 01' y es el modo de identificar el comienzo de un paquete PES.
- *Identificación de Flujo (1 byte).* Indica que tipo de datos hay en la carga útil del paquete, es decir, si los datos que transportan pertenecen a un flujo elemental de video, de audio o de datos.
- *Longitud del Paquete (2 bytes)*. Indican la longitud exacta del paquete permitiendo saber dónde termina y empieza el siguiente. Si estos dos bytes se ponen a cero el paquete puede exceder los 64KB que como longitud máxima tenía un PES.

A continuación de este encabezado, se transmite un **Encabezado PES Optativo** que se adapta a los requisitos del flujo elemental transmitido en ese instante.

- *Bits '10' (2 bits).* Este campo siempre vale lo mismo.
- *PES Scrambling Control (2 bits).* Define si el cifrado se usa y que método se emplea.
- *Prioridad del Paquete PES (1 bit).* Indica la prioridad del actual paquete PES.
- *Alineación de los Datos (1 bit)*. Indica si la carga útil empieza por el código de inicio de un flujo elemental.
- *Copyright (1 bit).*Indica si la carga útil está protegida con *Copyright.*
- *Paquete es Original o Copia (1 bit).* Indica si este es el flujo elemental original.

- *Flags (8 bits).* Indica que campos opcionales de esta cabecera están presentes en el paquete. Los Campos Opcionales en el encabezado optativo contienen, entre otras cosas, las Marcas de Tiempo de Presentación o PTS *(Presentation Time Stamps)* y las Marcas de Tiempo de Decodificación o DTS (*Decoding Time Stamps)* que son necesarias para la sincronización del video y del audio.
- *Longitud Encabezado PES (1 byte).* Indica la longitud total que tiene la cabecera del paquete PES.
- *Bytes de Relleno.* Al final del encabezado optativo puede haber allí unos bytes de relleno.

Siguiendo el encabezado del PES completo se transmite ***payload*** del flujo elemental.

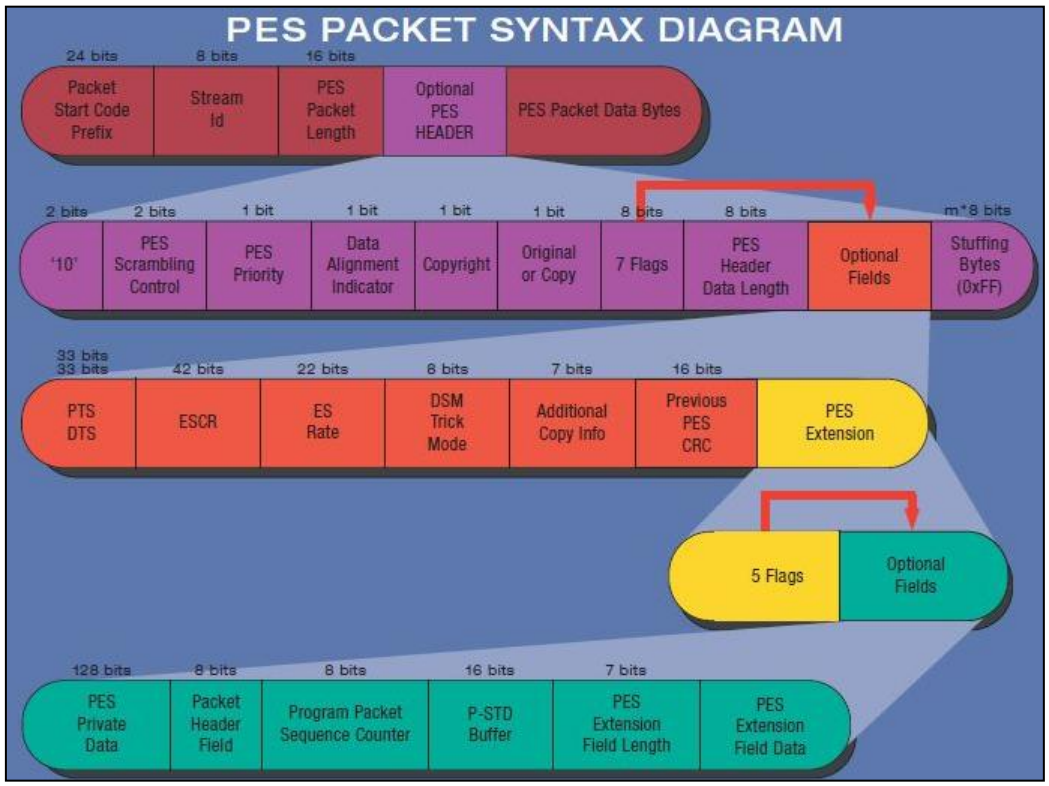


Figura 20. Sintaxis del Paquete PES

3.2.2.3. Multiplexación de los paquetes PES

Los paquetes PES son de longitud excesivamente grande para una buena transmisión, más si se tiene en cuenta que el objetivo es formar una única señal multiplexada de

datos MPEG-2 que transporte varios programas, en donde cada programa está formado por varios flujos elementales.

Para una mejor transmisión se dividen los paquetes PES en paquetes menores con una longitud constante de 188 bytes, en donde los 4 primeros bytes corresponden al encabezado y los siguientes 184 bytes son de carga útil. A estos paquetes se les conoce como paquetes de transporte *(Transport Packets)*. Un paquete de transporte solamente puede contener datos tomados de un único paquete PES, para ello hace uso, si es necesario, del campo de adaptación para rellenar el espacio sobrante. Este campo también se utiliza para enviar el Reloj de Referencia del Programa o PCR (*Program Clock Reference*) al decodificador para sincronizar su reloj con el reloj del programa. Debe aparecer en el flujo de transporte al menos una vez cada 0,1 segundos. En la figura 21 se muestra la formación de dichos paquetes.

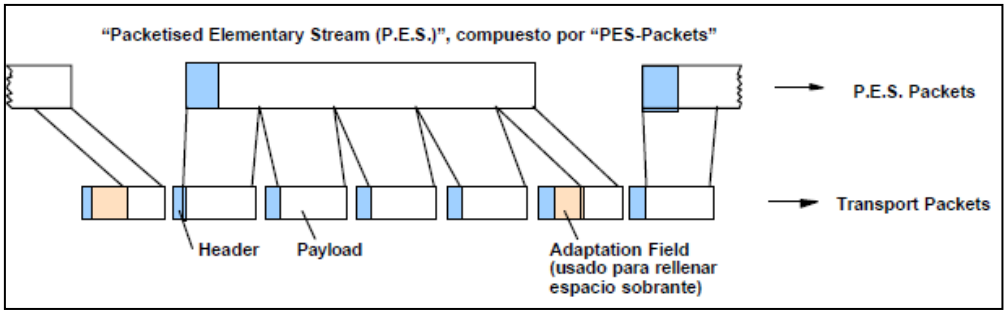


Figura 21. Formación de los paquetes de transporte

La sucesión de todos los paquetes de transporte pertenecientes a cada uno de los diferentes flujos PES, que a la vez pertenecen a los diferentes programas que están siendo transmitidos en ese momento, forman lo que se llama el Flujo de Transporte MPEG-2 o MPEG-2 TS *(MPEG-2 Transport Stream)*.

No hay un orden establecido a la hora de aparecer los paquetes en el multiplexor, es un servicio bajo demanda. El único requisito necesario es que los paquetes de transporte pertenecientes a un mismo PES sigan un orden cronológico.

Aparte de los paquetes destinados a transportar los ES de la señal, en el flujo de transporte viajan otros paquetes que contienen información sobre el servicio ofrecido

actualmente, así como paquetes “nulos” que se emplean para absorber eventuales reservas de capacidad del multiplexor.

La figura 22 muestra de forma simplificada como se lleva a cabo la conformación del Multiplex de Transporte TS.

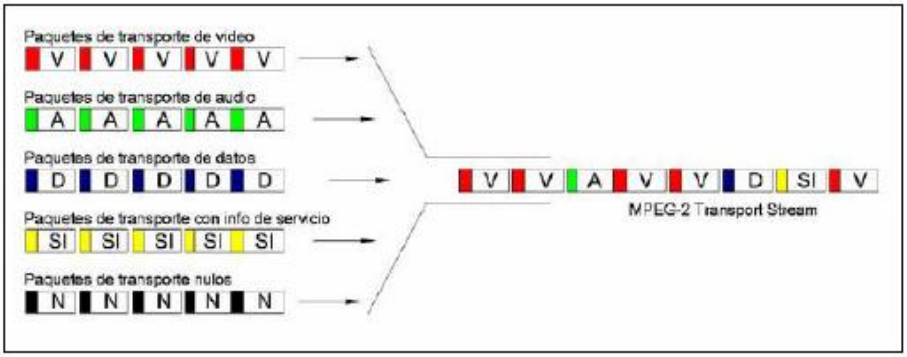


Figura 22. Conformación del Multiplex de Transporte TS

Estructura del paquete de transporte

El flujo de transporte MPEG-2 está formado por paquetes que tienen una longitud fija de 188 bytes, con 4 bytes de cabecera y 184 bytes de carga útil, figura 23. La carga útil puede contener video, audio o datos generales.

La **cabecera** de los paquetes de transporte tiene una longitud de 4 bytes.

- *Byte de Sincronización (1 byte)*. Sirve para lograr la sincronización del sistema con respecto al flujo de datos entrante. Gracias a su valor fijo de 0x47 y al tamaño constante de los paquetes se puede averiguar con facilidad el comienzo de cada paquete.
- *Bit de Indicador de Error (1 bit)*. Marca los paquetes de flujo de transporte como errados después de su transmisión.
- *Bit de Comienzo PES (1 bit)*. Si está activado indica que el primer byte de la carga útil del paquete coincide también con el primer byte de un paquete PES
- *Bit de Prioridad (1 bit)*. Sirve para dar prioridades a ciertos paquetes cuando sea necesario.
- *Identificador de Paquete* o *PID (13 bits).* Identifica los paquetes asociados a un determinado PES de entre todos los demás.

- *Control del Cifrado de Transporte (2 bits)*. Indica si hay o no datos cifrados en el *payload*.
- *Control del Campo de Adaptación (2 bits).*Indica si la cabecera tiene o no campo de adaptación.
- *Contador de Continuidad (4 bits).* El codificador lo incrementa en 1 cada vez que envía un paquete de la misma fuente. Esto permite que el decodificador sea capaz de deducir si ha habido una pérdida (o ganancia incluso) de un paquete de transporte y evitar errores que no se podrían deducir de otra manera.

Bajo ciertas circunstancias pude ser necesario transmitir una cabecera mayor de 4 bytes, así que se extiende la cabecera original en el campo de la carga útil, pero manteniendo el tamaño de paquete fijo en 188 bytes. Este encabezado extendido se denomina **Campo de Adaptación**.

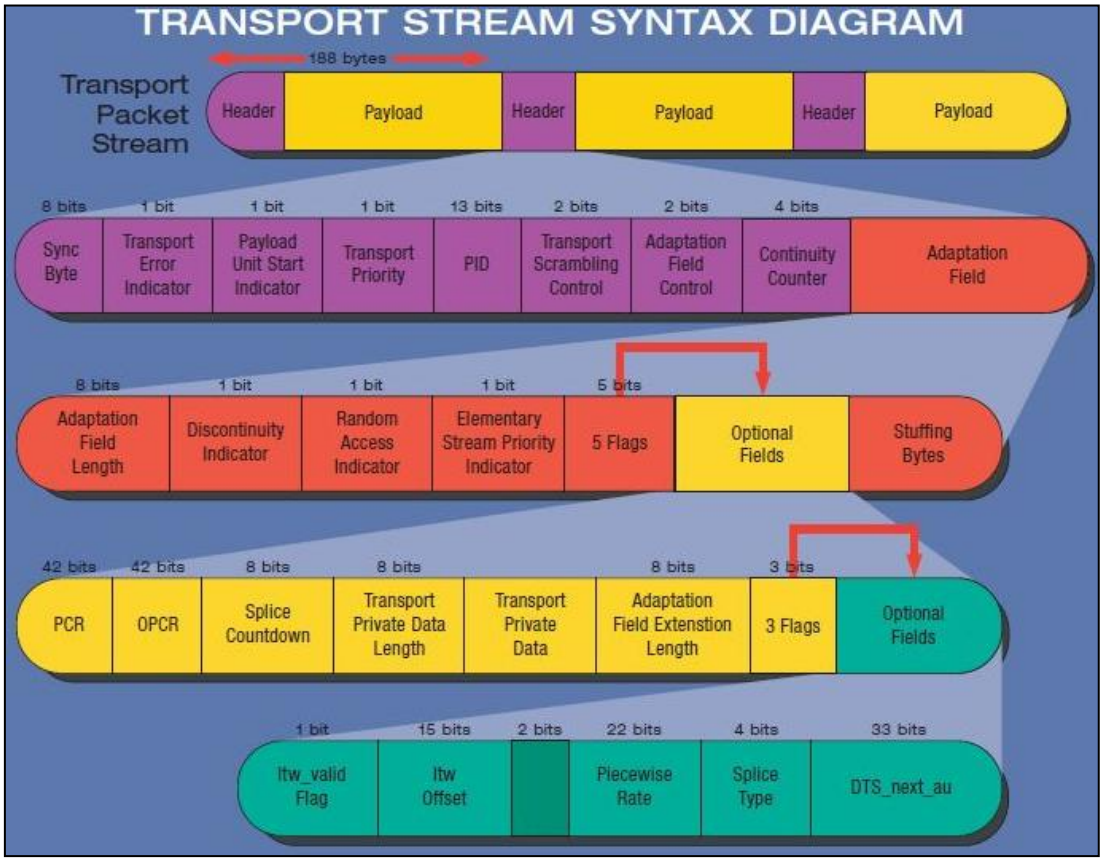


Figura 23. Sintaxis del Paquete de Transporte

3.2.3. Información específica de los programas (PSI)

Un múltiplex de transporte MPEG-2 puede transportar varios programas, cada uno de estos compuestos por uno o varios PES, de manera que para guiar y simplificar el proceso de demultiplexación y presentación de los programas en el decodificador, MPEG-2 define cuatro tipos de tablas de señalización que juntas constituyen la información específica de los programas PSI *(Program Specific Information*). Está definida por el MPEG para la Capa de Sistema (ISO/IEC 13818-1) y son:

- *Program Association Table* (PAT)
- *Conditional Access Table* (CAT)
- *Program Map Table* (PMT)
- *Private*

Cada tabla está formada por paquetes de datos que son reconocidos por un particular PID dentro del flujo de transporte.

Estas tablas se explican más detalladamente en la sección **Visualización en el Receptor,** en donde se muestra como el receptor hace usos de estas tablas, junto con las de Información de Servicio SI (*Service Informatcion*) propias de DVB, para presentar los programas**.**

3.3. DVB-S (*Digital Video Broadcasting by Satellite*)

El DVB-S es el estándar definido para la transmisión vía satélite de televisión digital creado por la organización europea DVB en 1995 y recogido en el estándar ETSI EN 300 421, especifica los procesos de codificación de canal y de modulación para un adecuado funcionamiento cuando se usan los canales de transmisión satelital. Como en el restos de estándares DVB, la señal de entrada normalizada es el flujo de transporte MPEG-2.

Dicho flujo de transporte, obtenido mediante el proceso conocido como Codificación de Fuente, es una adaptación del estándar MPEG-2 recogido en ISO/IEC 13818-1, que se estructura multiplexando varios programas.

Originalmente estaba pensado para la transmisión vía satélite de TV digital, pero luego fue adoptado para la transmisión de otros tipos de información debido a su sencillez y flexibilidad. Ha sido ampliamente aceptado en Europa y en casi todos los continentes, excepto en algunos países como Estados Unidos, Japón y Canadá donde coexisten con otros sistemas. Esto se recoge en la figura 24.

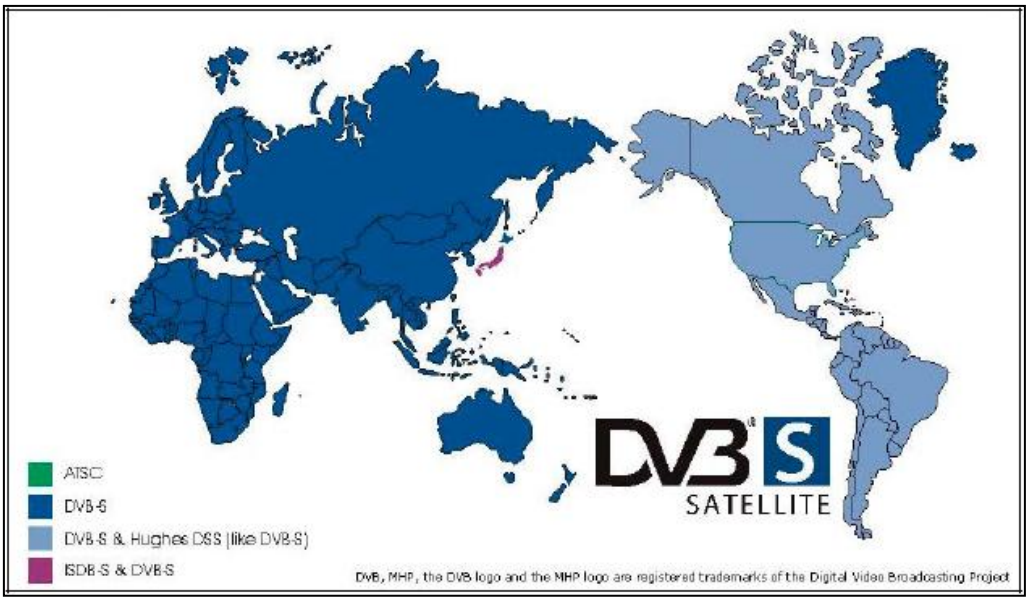


Figura 24.Estándares de TV digital por satélite

3.3.1. Sistema de transmisión DVB-S

En este apartado se explica como la señal es adaptada al medio de transmisión mediante la codificación de canal definida por el sistema de transmisión DVB-S.

3.3.1.1. Definición del sistema

El sistema está definido como aquel bloque funcional del equipo DVB-S que está encargado de la adaptación de las señales en banda base, entregadas por el múltiplex de transporte MPEG-2, a las características del canal satelital. El siguiente procedimiento debe ser aplicado al flujo de transporte antes de ser transmitido hacia el satélite:

- Adaptación del múltiplex de transporte y aleatorización para dispersión de energía.

- Codificación externa (*Reed-Solomon*).
- Entrelazado de la codificación interna (códigos convolucionales).
- Codificación interna (código convolucional perforado).
- Filtrado de banda base para la modulación.
- Modulación.

En la figura 25 se muestra el diagrama de bloques del sistema de transmisión que emplea éste estándar:

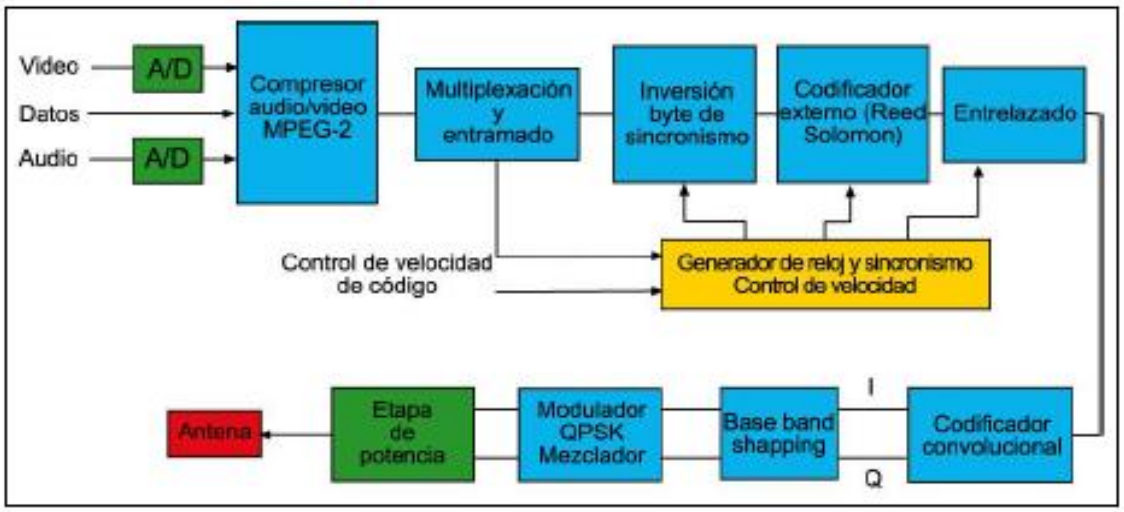


Figura 25. Diagrama de bloques funcional del sistema DVB-S

3.3.1.2. Parámetros del sistema

El método de modulación escogido fue QPSK (*Quaterny Phase Shift Keying*), ya que es el más apropiado para las comunicaciones por satélite, en donde la señal tiene que viajar grandes distancias y por lo tanto la señal sufre importantes atenuaciones, del orden de 200dB para un satélite que se encuentre en la órbita geoestacionaria. Se buscaba un método que fuera prácticamente inmune al ruido y que tuviera una amplitud constante para que la señal pudiera ser transmitida por el satélite cerca del punto de saturación de su HPA (*High Power Amplifier*). La clave es trabajar en este punto ya que es el lugar donde existe mayor eficiencia energética, hacerlo en otro punto significaría un desaprovechamiento de la energía, y en un satélite es un bien escaso.

Los satélites que pueden ser usados con DVB-S son aquellos con transpondedores con anchos de banda entre 26 y 36MHz. Por este motivo es necesario escoger una Tasa de Símbolo que produzca un espectro menor que el ancho de banda del transpondedor del satélite. Típicamente esta tasa de símbolo es de 27,5MS/s, con lo que usando una modulación QPSK nos da una tasa bruta de datos de 55Mbps.

$$R_b Bruto = R_S \cdot n^{\underline{o}} bits = 55 Mbps$$

Al ser una comunicación por satélite, hay que proporcionarle una buena protección frente a errores. Para ello se hace uso de códigos FEC (*Forward Error Correction*) concatenados. En primer lugar se tiene una un código de bloque *Reed-Solomon* que protege a la señal frente a ráfagas de errores, añadiendo 16 bytes de protección de error al paquete del flujo de transporte, obteniendo así un paquete de longitud de 204 bytes. Esto se conoce como codificación RS (204,188).

$$R_b Neto_{R-S} = R_b Bruto \cdot \frac{188}{204} = 50{,}69 Mbps$$

Esta protección contra errores no es suficiente por lo que se añade un segundo bloque codificador, en este caso un codificador convolucional que se inserta después del bloque *Reed-Solomon* para proteger a la señal de errores aleatorios. Todo esto hace que el flujo de datos aumente, pudiéndose controlar a través *del Code Rate* que dependerá de la eficiencia espectral y el comportamiento del canal. La relación de código se define como la relación entre la tasa de datos a la entrada del bloque convolucional y la tasa de datos a su salida. En DVB-S la relación de código se puede escoger entre 1/2, 2/3, 3/4, 5/6 y 7/8. En aquello casos en los que sea necesario una alta protección se elegirá una relación de código de 1/2, ya que por cada bit de entrada saldrán dos bits de salida, con lo que la tasa neta de datos será mínima. Normalmente una relación de código óptima es 3/4, obteniendo una tasa neta final de datos de 38,01Mbps.

$$R_b Neto_{DVB-S} = R_b Bruto \cdot \frac{188}{204} \cdot \frac{3}{4} = 38{,}01 Mbps$$

3.3.1.3. Codificación de canal

Una vez que se tiene el flujo de transporte MPEG-2 se debe realizar la codificación de canal, mediante la cual se añade suficiente redundancia y protección a la señal para hacerla más robusta con vistas a poder corregir los errores-*Forward Error Correction*- después de pasar por el canal de transmisión.

El flujo de datos de entrada está organizado en paquetes con una longitud fija de 188 bytes, en donde el primer byte es el byte de sincronismo con un valor fijado en 0x47.

La figura 26 muestra de manera esquemática los diferentes bloques por los que pasa la señal hasta que es radiada.

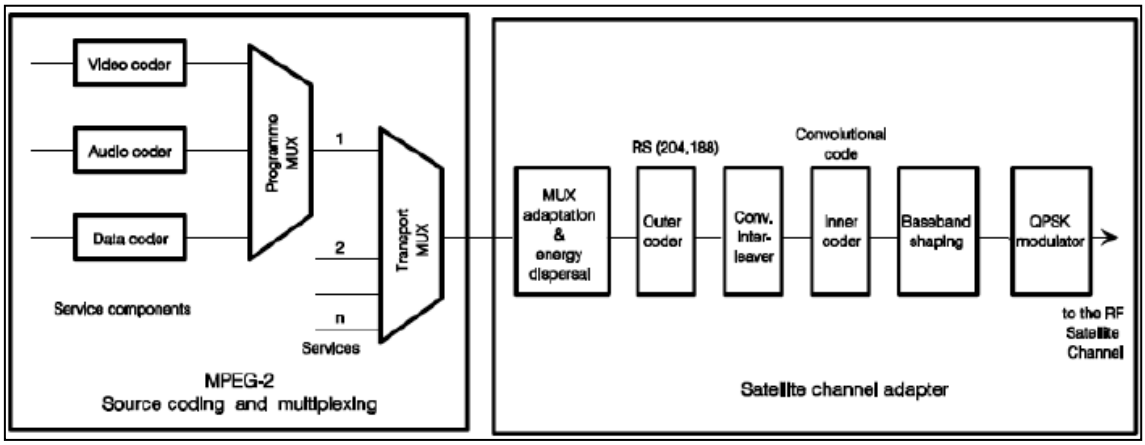


Figura 26. Codificación de canal de DVB-S

Adaptación y Dispersión de Energía

La Dispersión de Energía consiste en la cuasi-aleatorización de la señal de entrada para obtener un espectro más equilibrado, es decir, que la potencia se reparta por igual en todo el ancho de banda disponible, evitando que se transmitan una sucesión larga de unos o ceros seguidos. Esto no se desea ya que no contienen ninguna información de reloj o causa líneas espectrales discretas sobre un periodo particular. Para llevar a cabo esta tarea, el multiplex de entrada pasa por un proceso cuyo esquema se muestra en la figura 27.

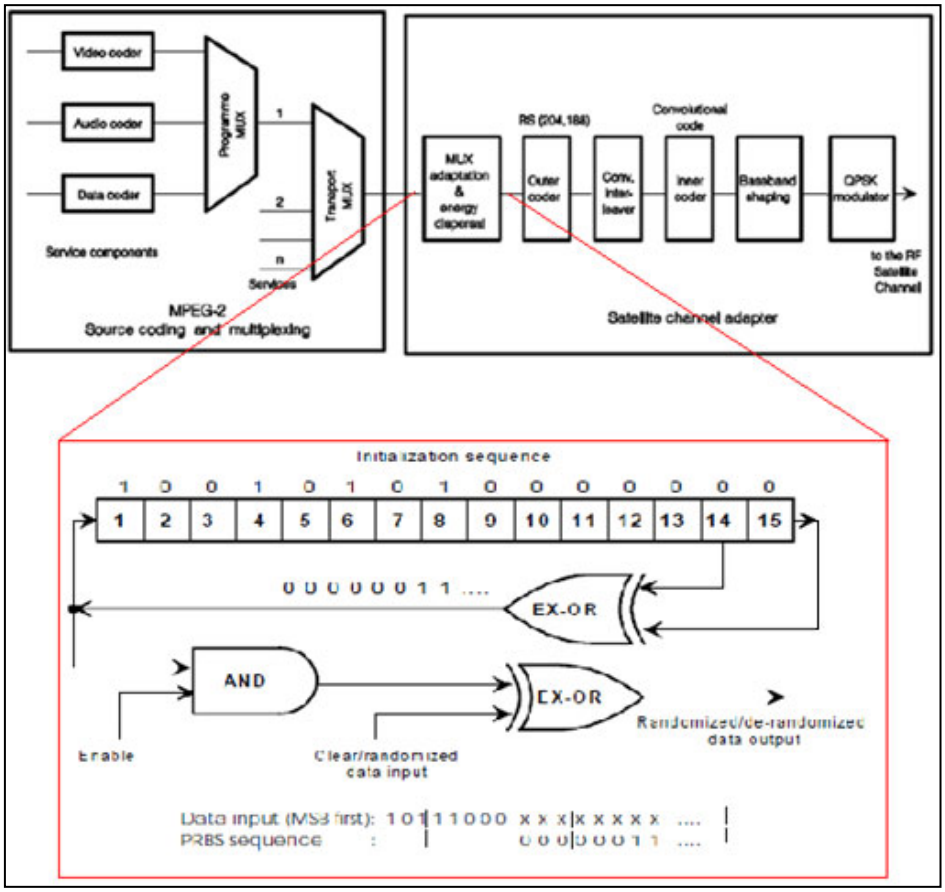


Figura 27. Aleatorizador DVB-S del flujo de transporte

En este proceso se trata de obtener una secuencia binaria pseudoaleatoria PRBS, para lo cual se emplea un generador que usa el polinomio:

$$1 + x^{14} + x^{15}$$

El generador PRBS, que sirve tanto para desordenar como para ordenar, está compuesto por un registro de 15 posiciones que se carga con la secuencia '100101010000000', el cual debe iniciarse al comienzo de un conjunto de 8 paquetes de transporte. La aleatorización se consigue haciendo una operación lógica XOR entre la señal original y la secuencia binaria obtenida del generador, en donde los bytes de sincronización no se ven afectados.

Para proporcionarle una señal de inicialización al decodificador receptor que le indique cuando debe inicializar el generador, es decir, que identifique cuando se han recibido 8 paquetes aleatorizados, el byte de sincronización del octavo paquete enviado por el

transmisor se invierte, pasando de tener un valor inicial de 0x47 a 0xB8. Esto es lo que se conoce como Adaptación del Multiplexor.

Codificación *Reed-Solomon* (Codificación Externa)

Para permitir la corrección de errores FEC en la recepción, se introduce una cierta redundancia en la estructura de los paquetes de transporte, es decir, se produce una Codificación. A diferencia del proceso de aleatorización, la codificación también es aplicada al byte de sincronismo.

En todos los estándares DVB se emplea una Codificación Externa, y solo en el caso de los estándares de transmisión vía satélite y terrestre se emplea una codificación extra conocida como Codificación Interna para darle una mayor robustez a la señal, debido a que son medios en los que hay una alta probabilidad de error.

La codificación elegida es la codificación *Reed-Solomon* (204,188, t=8), en donde se introducen 16 bytes de paridad a cada paquete del flujo de transporte. Con esto el decodificador es capaz de corregir hasta 8 bytes erróneos por cada paquete de 204 bytes recibido.

En la figura 28 se muestra la estructura de los paquetes de transporte después de la "codificación externa".

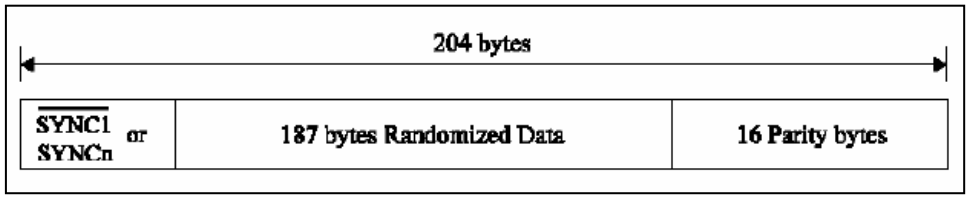


Figura 28. Paquete TS protegido por Reed-Solomon RS (204, 188,8)

Entrelazado (*Interleaving*)

Para evitar ráfagas de errores consecutivas los paquetes son entrelazados con el objetivo de dispersar a lo largo del tiempo las ráfagas de errores introducidas por el canal.

La figura 29 muestra el esquema de concepto del Entrelazado Convolucional, relativo al byte y con profundidad I=12, a que se someten los paquetes de transporte una vez han sido protegidos por la codificación externa *Reed-Solomon*.

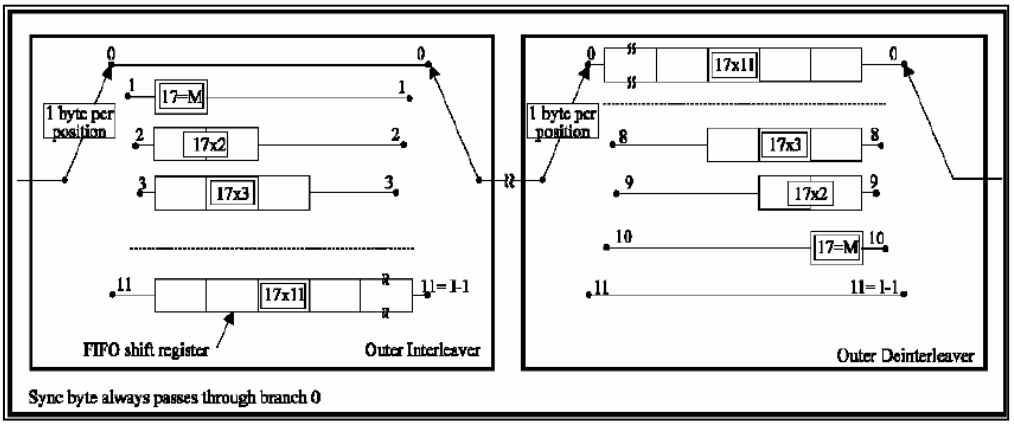


Figura 29. Entrelazado y Desentrelazado convolucional en DVB-S

Cada byte del paquete de 204 bytes se introduce de forma sucesiva en los 12 registros por los que está formado el Entrelazador (es decir: en el primer registro irán los bytes 1, 13, 25… en el segundo registro irán los bloques 2, 14, 26, etc.). Los bytes de sincronización siempre van por la rama 0 del Entrelazador para poder ser localizados. Una vez hecho esto, se forma la nueva trama, concatenando los contenidos de los registros. Las ráfagas de errores del canal afectarán a bytes sucesivos de la nueva trama, pero al deshacer el entrelazado se repartirán a lo largo de la trama original. Así se aumenta la eficiencia de la decodificación *Reed-Solomon*, ya que al llegar los errores más separados es más probable que los pueda corregir.

En la figura 29 se muestra como el paquete de entrada del flujo de transporte va sufriendo variaciones según va pasando por los distintos bloques.

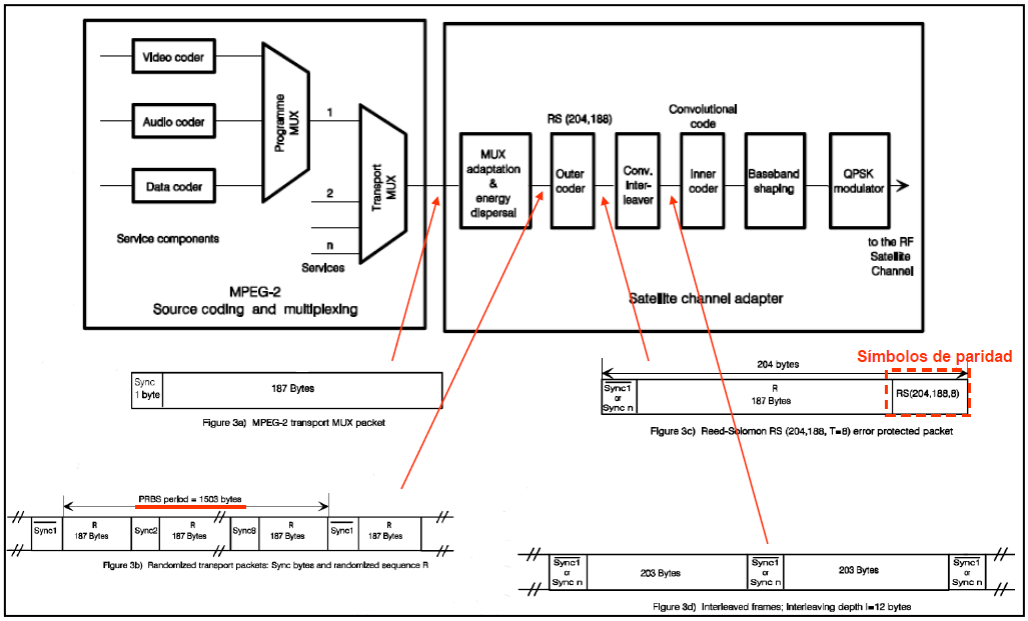


Figura 30. Entramado DVB-S

Codificación convolucional (Codificación Interna)

La protección contra errores proporcionada por la codificación *Reed-Solomon* no es suficiente, por lo que se añade un nuevo bloque para hacer la señal más robusta. Si bien el entrelazado preparaba a la señal para posibles ráfagas de errores, la codificación interna lo hace frente a errores aleatorios. Este bloque es fácilmente controlable a través del C*ode Rate*, el cual permite elegir una tasa u otra en función de las características del medio.

El *Code Rate* depende de la eficiencia espectral y del medio al que se vaya a transmitir la señal. Así se puede escoger entre 1/2, 2/3, 3/4, 5/6 y 7/8.

Como se ha dicho anteriormente, cuando se estaba calculando el régimen binario al que normalmente funciona el sistema DVB-S, una tasa de código optima sería 3/4. En donde por cada 3 bits de entrada salen 4 bits.

Filtrado en banda base

A la salida del convolucional se tienen las componentes en fase y cuadratura de la señal en banda base, por lo que antes de enviar la señal al modulador para su transmisión se pasa la señal por un filtro en banda base que lo que hace es limitar las componentes espectrales de la señal y así evitar la interferencia entre símbolos.

Esto se consigue con un filtrado en coseno alzado tipo **Nyquiest** con un factor de roll-off del 0,35%. Ahora el ancho de banda de la señal será 1,35 veces el ancho de banda de la señal ideal.

Modulación QPSK

Una vez que ya se tiene lista la señal para ser transmitida, solo hace falta adaptarla al medio por el cual va a ser enviada. Según las características del canal satelital, alta probabilidad de error y grandes atenuaciones, la modulación más apropiada es QPSK. Se trata de una modulación de amplitud constante en donde la información de la moduladora va insertada en fase de la portadora, por lo que la hace muy robusta frente a ruidos atmosféricos. Además tiene una eficiencia espectral muy alta y ocupa un ancho de banda reducido. La figura 31 muestra el proceso de adaptación de la señal.

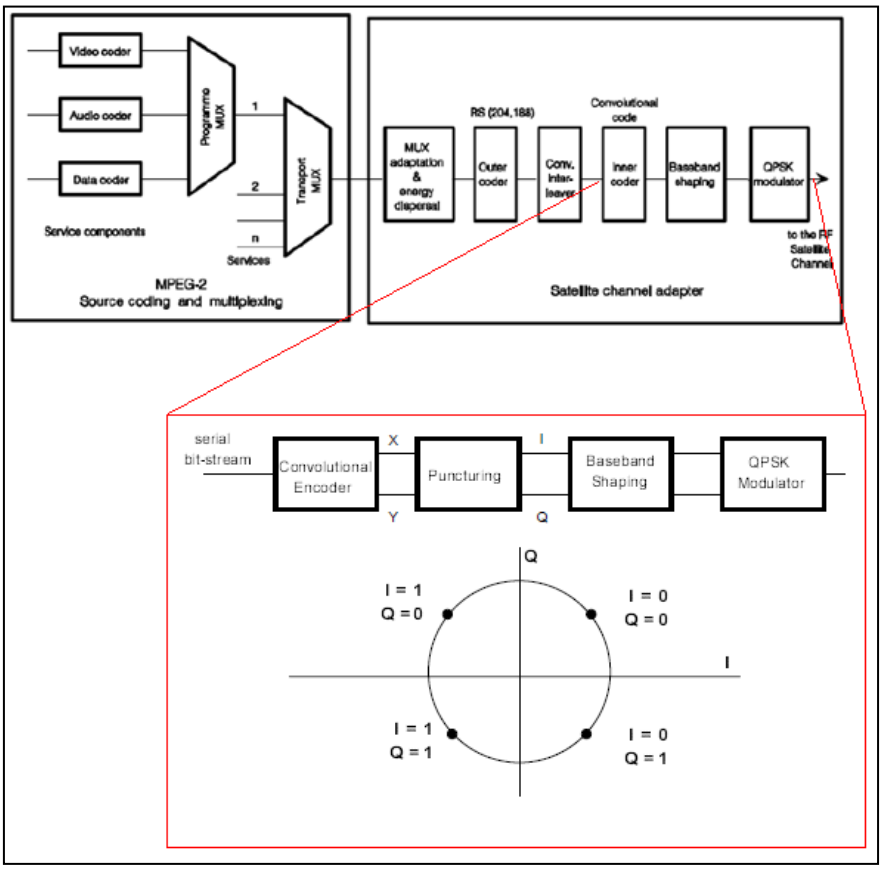


Figura 31. Codificación Interna y Modulación QPSK en DVB-S

Todo este proceso de adaptación de la señal para su transmisión se realiza de manera inversa en el receptor con el objetivo de obtener la señal original, es decir, el flujo de transporte MPEG-2 que se tenía a la entrada del transmisor.

3.3.2. Visualización en el receptor

En este apartado se va a tratar de explicar cómo el receptor es capaz de demultiplexar y presentar todos los programas que hay dentro de un flujo de transporte MPEG-2. Para ello se muestran los pasos que sigue desde que la señal llega al receptor hasta que está lista para su entrega. La figura 32 presenta esquemáticamente todos estos pasos.

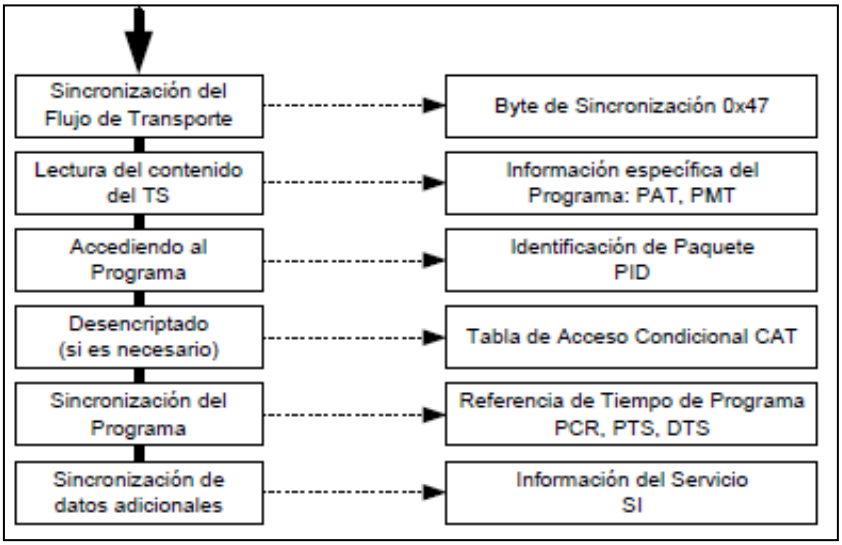


Figura 32. Visualización de los programas en el receptor DVB-S

1º Sincronización del Flujo de Transporte

Cuando la entrada del decodificador MPEG-2 se conecta a un flujo de transporte, lo primero que hace es “engancharse” al mismo, a la estructura de paquete, buscando los bytes de sincronización dentro del flujo de transporte. Estos bytes tienen siempre un valor de 0x47 y aparecen en cada uno de los paquetes del flujo de transporte a intervalos fijos de 188 bytes, que es el tamaño de cada paquete. Con estos dos factores, el valor fijo y el intervalo de presentación, el decodificador es capaz de realizar la sincronización. Si aparece un byte que tiene un valor de 0x47, el

decodificador examinará las posiciones “n” veces 188 bytes antes y después de este byte en el flujo de transporte en busca de la presencia de otro byte de sincronización. El valor 0x47 no es exclusivo de la cabecera y podría también aparecer en el campo de la carga útil, pero al no presentarse a intervalos fijos de 188 bytes, sería desechado como byte de sincronización.

2º Lectura del contenido deseado

Una vez que el decodificador MPEG-2 se encuentra sincronizado con el flujo de transporte, busca los paquetes que forman la denominada Información Específica del Programa o PSI *(Program Specific Information)*. La PSI describe la estructura instantánea del flujo de transporte y la relación existente entre los PIDs y los diferentes programas del flujo. Esto ayuda al decodificador a demultiplexar y presentar los programas en el receptor.

La Información Específica del Programa está formada por cuatro tablas de señalización que se transmiten periódicamente. Cada una de ellas está formada por paquetes de datos que son reconocidos por un particular PID dentro del flujo de transporte.

Program Association Table (PAT)

La Tabla de Asociación de Programa (PAT) es de uso obligatorio e indica la cantidad de programas que hay en ese momento en un flujo de transporte. Se transmite periódicamente, a intervalos de 0.5 segundos. Los paquetes que transportan estas tablas tienen un PID de 0x0000 para su fácil identificación dentro de la gran cantidad de paquetes que componen el flujo. En la parte de carga útil de la Tabla de Asociación de Programa, se transmite una lista de PIDs especiales, uno por cada programa que se esté transmitiendo. Estos PIDs son punteros a otras tablas llamadas Tablas de Mapeo de Programa o PMT *(Program Map Tables)*, en donde se describe cada programa individualmente señalando los flujos que lo forman. La figura 33 representa la estructura de un paquete perteneciente al PAT.

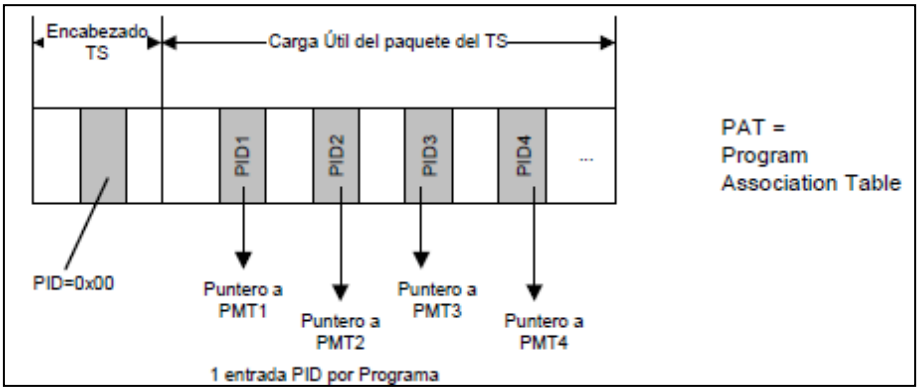


Figura 33. PAT (*Program Association Table*)

Program Map Table (PMT)

Las Tablas de Mapeo de Programa (PMT) están formadas por paquetes del flujo de transporte en donde la carga útil son los PIDs de los flujos elementales que forman un programa. Cada programa dentro del flujo de transporte tiene una tabla PMT asociada con él. Según MPEG-2, las tablas PMT pueden ser transportadas por paquetes con valores de PID arbitrarios, exceptuando los valores 0x0000 y 0x0001, que están reservados para las tablas PAT y CAT *(Conditional Access Table)* respectivamente. La figura 34 representa la estructura de los paquetes que la forman.

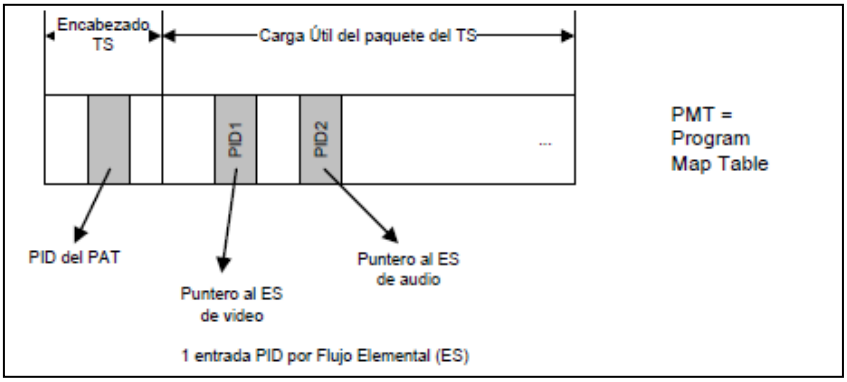


Figura 34. PMT (*Program Map Table*)

3º Acceso a un programa

A continuación, para acceder a un programa basta con capturar todos los paquetes del flujo de transporte con los PIDs indicados en la PMT correspondiente y desechar los otros. El decodificador se encarga de demultiplexar estos paquetes para formar los

nuevos paquetes PES. Si los flujos elementales no están cifrados, estos pueden ser decodificados directamente por el decodificador MPEG-2.

4º Acceso a programas cifrados (Acceso Condicional)

El sistema de Acceso Condicional (CA) sirve para limitar el acceso al contenido a ciertos usuarios mediante el cifrado de la información. Su estructura está definida por las especificaciones de la norma, pero los algoritmos de cifrado y descifrado son propios de cada proveedor. En el caso del DVB-S se emplea el *Common Scrambling Algorithm* (CSA) desarrollado por el DVB.

Toda la información necesaria para acceder a un programa encriptado es enviada a través del flujo de transporte por medio de los mensajes de acceso condicional: EMMs *(Entitlement Management Message)* y ECMs (*Entitlement Control Message*). Los EMMs se encargan de la administración de los derechos de cada receptor para acceder a un servicio, mientras que los ECMs contienen la *Control Word* encriptada.

Para localizar los flujos EMMs y ECMs dentro del flujo de transporte se incorpora al multiplex la Tabla de Acceso Condicional (CAT), identificada con el PID 0x0001. En la figura 35 se muestra de una forma simplificada la estructura de un paquete correspondiente al CAT.

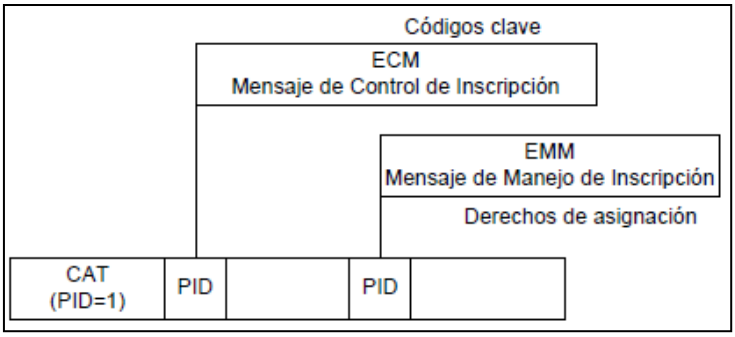


Figura 35. CAT (*Conditional Access Table*)

5º Sincronización del programa

Para la decodificación del audio y del video, es necesario unos pasos adicionales de sincronización. Para ello se emplea el PCR *(Program Clock Reference)*, incorporado en el encabezado opcional del paquete de transporte, el DTS *(Decoding Time Stamps)* y el

PTS *(Presentation Time Stamps)*, estos dos últimos transmitidos en el encabezado del paquete PES.

6º Sincronización de datos adicionales

DVB ha definido una serie de tablas con el objetivo de proveer al terminal receptor de la información suficiente de la red para que pueda operar en ella de una manera más simple. Estas tablas forman la Información del Servicio SI *(Service Information)* definida en el documento de la ETSI EN 300 468.

Tanto las tablas que forman el PSI de MPEG como las que forman el SI de DVB se transmiten dentro del flujo de transporte para ayudar al receptor en su tarea de representar los programas. A diferencia de las tablas del PSI, que solo suministran información sobre el flujo de transporte en las que están ubicadas, las tablas del SI también pueden suministrar información de servicios y eventos transportados por otros TS e incluso TSs transmitidos por otras redes. Ambas son identificadas dentro del flujo de transporte a través de sus PIDs. En la Tabla 1 se muestra las principales tabla que forman el SI.

Tablas Obligatorias	Tablas Opcionales
Network Information Table (NIT)	*Bouquet Association Table (BAT)*
Service Description Table (SDT)	*Running Status Table (RST)*
Event Information Table (EIT)	*Time OffsetTable (TOT)*
Time & Date Table (TDT)	*Stuffing Tables (ST)*

Tabla 1. Tablas del SI (*Service Information*)

Estas son tablas definidas por un agente externo al MPEG-2, por lo tanto tienen que seguir las normas establecidas para el uso de tablas privadas dentro del flujo de transporte. Estas normas dictan lo siguiente:

- Una tabla se transmite en la parte de la carga útil de uno o más paquetes del flujo de transporte con un PID especial que es reservado solo para esta tabla o algunos tipos de tabla.

- Cada tabla empieza con una IDentificación de tabla que es un byte especial que identifica exclusivamente a esta tabla.
- Cada tabla está compuesta por una o varias secciones, conocidas como sub-tablas y tienen un valor máximo de 4096 bytes. Las secciones en DVB están limitadas a 1024 bytes, excepto para las secciones que pertenecen a la EIT (*Event Information Table*) que pueden alcanzar los 4096 bytes.

A continuación se pasa a hacer una breve descripción de cada una de las tablas que forman la Información de Servicio de DVB.

***Network Information Table* (NIT)**

La NIT provee toda la información relacionada con los parámetros físicos de la red necesarios para transmitir el flujo de transporte, como por ejemplo frecuencia del canal, detalles del transpondedor del satélite, tipo de modulación, etc. Toda esta información queda almacenada en la memoria no volátil de los receptores para minimizar el tiempo de acceso cuando se cambia entre canales.

Esta tabla constituye el programa nº 0 y es transmitida en paquetes identificados con PID=0x0010.

***Service Description Table* (SDT)**

La SDT describe los servicios contenidos en un flujo de transporte en particular. Los servicios pueden ser parte del actual TS o parte de otro TS, los cuales son identificados por medio del *Table_Id*. A diferencia de las tablas PAT, esta contiene información de texto para el usuario como por ejemplo nombres de los servicios, nombre del proveedor y demás parámetros relacionados a los servicios.

Esta tabla es transmitida en paquetes identificados con PID=0x0011.

***Event Information Table* (EIT)**

La EIT contiene datos que hacen referencia a los acontecimientos que ocurren y que ocurrirán sobre el múltiplex MPEG recibido en la actualidad, y ocasionalmente sobre

otros múltiplex MPEG, como por ejemplo la denominación, hora de comienzo, duración, etc.

Esta tabla es transmitida en paquetes identificados con PID=0x0012.

***Time & Date Table* (TDT)**

La TDT solo contiene la hora UTC (*Coordinated Universal Time*) y fecha actual. Sirve para configurar el reloj interno del receptor. Está formado por una única sección.

Esta tabla es transmitida en paquetes identificados con PID=0x0014.

***Bouquet Association Table* (BAT)**

La BAT proporciona información acerca de los *bouquets* y los servicios que en él se agrupan, entendiéndose como *bouquet* a una colección de servicios disponibles para el usuario que pueden pertenecer a redes diferentes.

Esta tabla es transmitida en paquetes identificados con PID=0x0011. Como se aprecia tiene el mismo identificador que la SDT ya que la información que transportan es la misma, con la diferencia que el SDT es para un solo canal físico y el BAT es para un conjunto de canales físicos. Se diferencia en el *Table_Id.*

***Running Status Table* (RST)**

La RST actualiza de forma rápida la información relativa a un evento, ya esté sucediendo o no en ese instante. Se transmiten únicamente cuando un hay un cambio, a diferencia del resto de tablas que se envían periódicamente. En el momento que se transmite una RST para actualizar el estado de un evento, invalida el estado de ese evento, transmitido por la tabla EIT anteriormente. La siguiente vez que se transmita la EIT debe de contener los bits de estado actualizados.

Esta tabla es transmitida en paquetes identificados con PID=0x0013.

***Time OffsetTable* (TOT)**

La TOT proporciona información relativa a la fecha y hora real así como la diferencia con respecto a la hora local.

Esta tabla es transmitida en paquetes identificados con PID=0x0014.

***Stuffing Tables* (ST)**

Estas tablas de relleno se emplean para invalidar tablas que ya no sirven. Por ello usan paquetes que comparten valores de PID con otros tipos de tablas, tales como: 0x0010; 0x0011; 0x0012 0x0013 y 0x0014.

3.3.3. Inserción de datos en el flujo de transporte

Hasta este punto se ha visto la manera de transportar un programa a través del flujo de transporte MPEG-2. Ahora se verán los diferentes métodos que emplea el sistema DVB para transportar distintos tipos de datos dentro del TS para construir las redes de comunicación VSAT con DVB-S.

El transporte de datos se refiere a la descarga de software desde el satélite, el empleo de servicios por internet, la televisión interactiva, el envío de datos a estaciones remotas, etc.

Estos métodos están recogidos en los estándares ETSI EN 301 192 y ETSI TR 101 202, los cuales son:

- "*Data Piping*"
- "*Data Streaming*"
- "*Data Carousel*"
- "*Object Carrousel*"
- "*Multiprotocol Encapsulation*"

Cada uno de estos mecanismos tiene unas características especiales en relación con el filtrado, coste, tamaño… que le hacen más apropiado para transportar un tipo u otro de datos. En la figura 36 se muestra esquemáticamente la idea general.

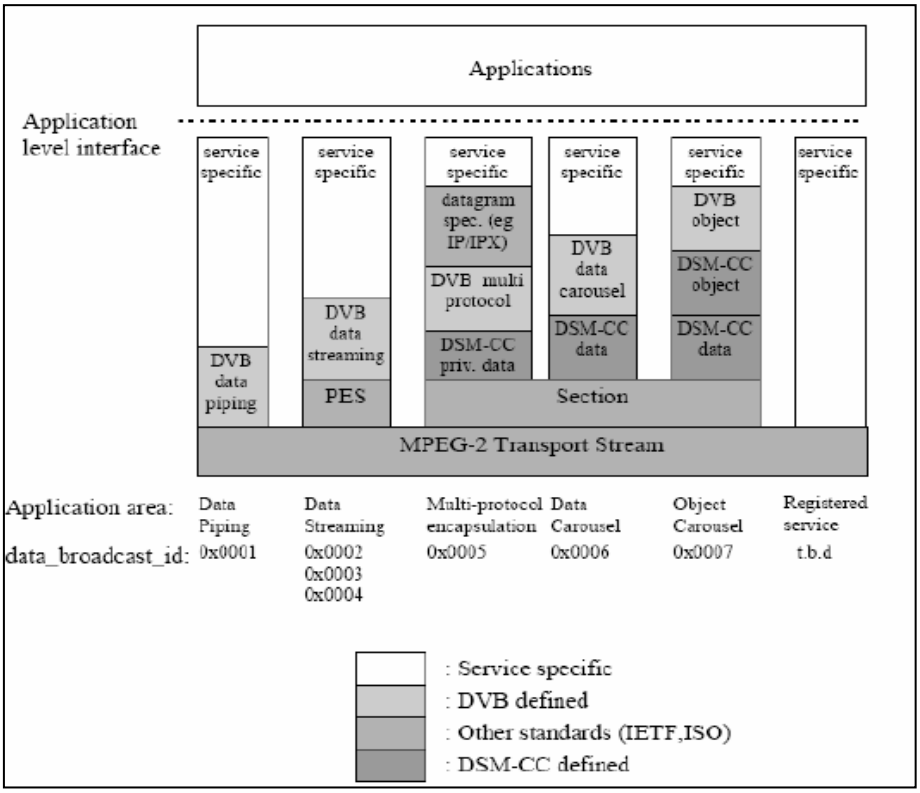


Figura 36. Métodos para transportar datos sobre el flujo MPEG-2

Generalmente los datos de cualquier aplicación son transmitidos a las capas inferiores de la pila de protocolos como paquetes de longitud variable. En cambio, en el flujo de transporte el tamaño del paquete es fijo, por lo que es necesaria la fragmentación de los datos. La fragmentación se puede realizar por tres métodos:

- Mecanismos privados basados en Data Pipe.
- Flujo Elemental Paquetizado de MPEG-2 (PES).
- Secciones MPEG-2.

Si los datos no están empaquetados o el método de empaquetado es irrelevante para la cadena de transmisión DVB, el método más apropiado es el *Data Pipe*.

Cuando se desea una transmisión síncrona, sincronizada o asíncrona, el método más adecuado es el basado en paquetes PES. Este método permite sincronizar perfectamente diferentes cadenas de datos, tal y como hace MPEG para sincronizar los paquetes de audio y video de los programas del flujo de transporte.

Las secciones MPEG-2 también pueden ser usadas para transportar paquetes de datos a través del flujo de transporte. En este caso la transmisión es asíncrona y el tamaño

de la sección es variable, fijando una longitud máxima de 4 KB. La sección MPEG-2 se construye de tal manera que el demultiplexor en el receptor es capaz de seleccionar una única sección mediante hardware, lo cual disminuye la carga de software. Este es el principal motivo por el que se elige este mecanismo para la transmisión de protocolos encapsulados y carruseles de datos.

3.3.3.1. Data piping

El *Data Pipe* es un mecanismo simple de transporte asíncrono en donde los datos son insertados directamente en la carga útil de los paquetes de transporte y no hay una relación de tiempo entre los datos y el resto del contenido. La relación se establece de extremo a extremo y no necesita de ningún protocolo intermediario.

El canal puede ser visto como un "tubo", figura 37, en donde los datos son introducidos por uno de los extremos y recogidos en el otro extremo. Para que el receptor sea capaz de entender el flujo de datos que le llega, es necesario que antes de la transmisión haya un dialogo entre el transmisor y el receptor, en donde por ejemplo se indique como los datos han sido distribuidos a lo largo de los 188 bytes del paquete de transporte.

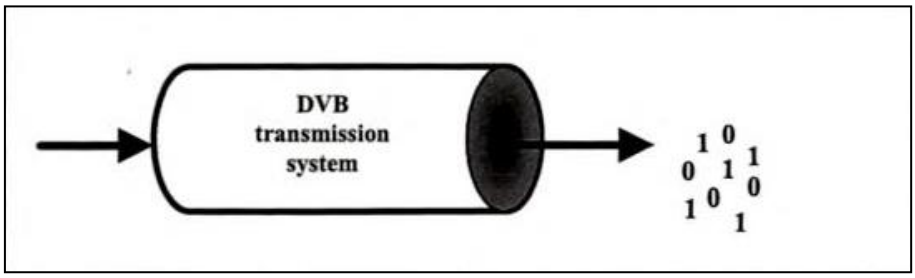


Figura 37. *Data Piping*

Data Piping prácticamente no se usa mucho, pero todavía existe en algunos entornos de emisión controlado por agentes propietarios.

3.3.3.2. Data streaming

El *Data Streaming*, figura 38, es un mecanismo de transporte extremo a extremo basado en la encapsulación de los datos en paquetes PES. Se puede dividir en tres áreas de trabajo según los requisitos y necesidades de los datos a transmitir.

Asynchronous Data Streaming

En este caso no es necesaria una sincronización entre los datos. Es un mecanismo similar al *Data Pipe* salvo porque aquí los datos están empaquetados dentro de segmentos de 64KB. Estos segmentos son insertados en paquetes PES en donde posteriormente son divididos en unidades de 184 bytes con el objetivo de encajar dentro del *payload* de los paquetes de transporte MPEG-2. Un posible uso para este mecanismo es el envío de grandes ficheros.

Synchronous Data Streaming

Se emplea en los casos en los que hay una tasa de datos constante. Una tasa de datos constante se emplea para transportar el reloj de referencia del codificador, con el cual poder ajustar el reloj del receptor.

Synchronised Data Streaming

Algunas aplicaciones necesitan que diferentes flujos de datos este sincronizados entre sí. El ejemplo más típico es la relación que se establece entre el flujo de video y el flujo de audio de un programa. Es necesario asegurar que ambos estén completamente sincronizados para una correcta representación.

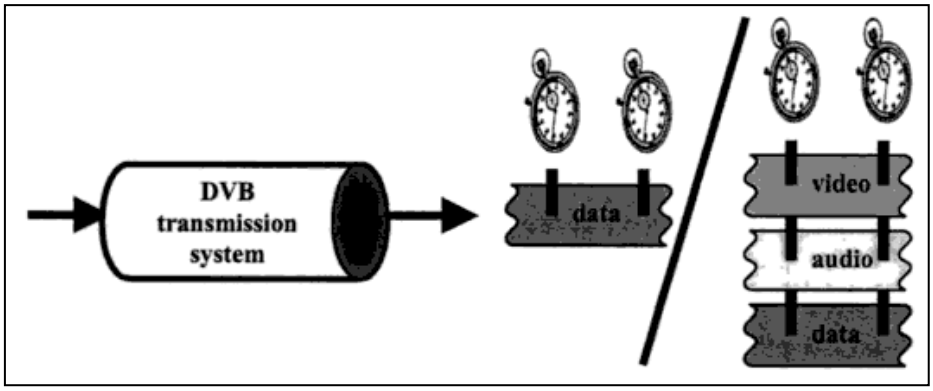


Figura 38. *Data Streaming*

3.3.3.3. Data carousel

El *Data Carousel* consiste en la transmisión periódica de archivos de datos a través de la red DVB. Estos archivos son de tamaño conocido y pueden ser actualizados, añadidos o eliminados del carrusel de datos en cualquier momento. Usa como unidad de paquete las secciones de tamaño fijo DSM-CC (*Digital Storage Media Command and Control*). Se emplea para el envío de las guías electrónicas de programas EPG y el teletexto en los medios de difusión de TV.

3.3.3.4. Object carousel

El *Data Carousel* funciona bastante bien para sistemas de teletexto o aplicaciones simples en donde no es importante conocer la estructura de los datos ni el contenido de los mismos, siendo el receptor el encargado de conocer dicha información. Para aplicaciones más importantes la solución es el *Object Carousel*. Este se construye sobre el *Data Carousel* añadiendo información sobre la estructura de los datos que están siendo transmitidos a través de incorporar los nuevos conceptos de archivos, directorios y flujos.

3.3.3.5. Multiprotocol encapsulation (MPE)

El *Data Carousel* y el *Object Carousel* permiten a los transmisores transportar datos a los receptores usando secciones MPEG-2. Para algunos tipos de datos como archivos

simples esto está bien, pero no es recomendable para todos los tipos de datos que pueden ser transmitidos a un decodificador. Para el caso particular del tráfico en Internet los carruseles no son prácticos.

Aquí es donde interviene la encapsulación multiprotocolo DSM-CC que permite a los sistemas transportar datos IP sobre secciones MPEG-2, proporcionando a el decodificador una conexión simple de una sola dirección con Internet.

IP emplea una combinación de dirección IP y dirección MAC *(Media Access Control)* para localizar al receptor. Gracias a los buenos resultados obtenidos, MPE usa el mismo sistema. Cada receptor tiene una dirección MAC formada por 48 bits que le identifica inequívocamente a fin de que cada paquete pueda ser entregado al destinatario correcto. Sin embargo, DVB no especifica la manera en la que son asignadas dichas direcciones.

Una dirección MAC por sí sola no es de gran utilidad, se necesita asociarla a una dirección IP. Esto se consigue con la incorporación de una nueva tabla al *Service Information* de DVB, la INT (*IP/MAC Information Table*).

Las secciones MPE pueden transportar cualquier paquete de transporte de la capa 3 del protocolo OSI. No obstante, debido a su importancia, ha sido optimizado para transportar paquetes IP. Estas secciones siguen el formato normal para las secciones privadas DSM-CC, pero están modificadas para facilitar al receptor el filtrado de los paquetes según la dirección MAC a través de hardware.

La seguridad en este tipo de redes es muy importante, así la encapsulación permite la transmisión segura de los datos a través de la encriptación de los paquetes y el cambio dinámico de direcciones MAC.

Transporte de datos

La encapsulación MPE consiste básicamente en tomar los datagramas de datos de las capas superiores, normalmente serán datagramas IP, empaquetarlos en secciones MPE añadiendo una cabecera MPE y un *checksum* o CRC_32 (*Cyclic Redundancy Check*) para el control de errores al final de cada datagrama, y posteriormente transmitirlos dentro

de los paquetes de transporte MPEG. La figura 39 muestra visualmente como se lleva a cabo la encapsulación multiprotocolo.

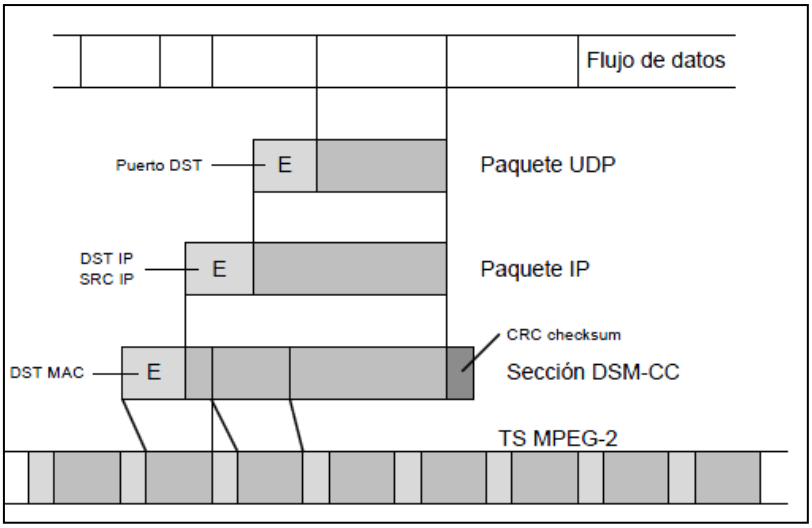


Figura 39. Encapsulación Multiprotocolo (MPE)

El formato de la sección permite el fragmentar los datagramas en múltiples secciones. Así si la longitud del datagrama es superior a 4080 bytes (incluida la posible cabecera LLC/SNAP), será necesaria la fragmentación del datagrama.

En el caso concreto de IP, los datagramas solo se pueden transportar en una única sección, por lo que habrá que tener en cuenta la longitud máxima de los mismos. Así, para el caso en el que la cabecera LLC/SNAP está omitida, se configurará la MTU *(Maximum Transfer Unit)* del datagrama en 4080 bytes o menos para evitar la fragmentación del datagrama. En cambio, si dicha cabecera está presente, la MTU se configura en 4074 bytes o menos. La figura 40 muestra la formación de los paquetes del flujo de transporte MPEG partiendo de un datagrama IP.

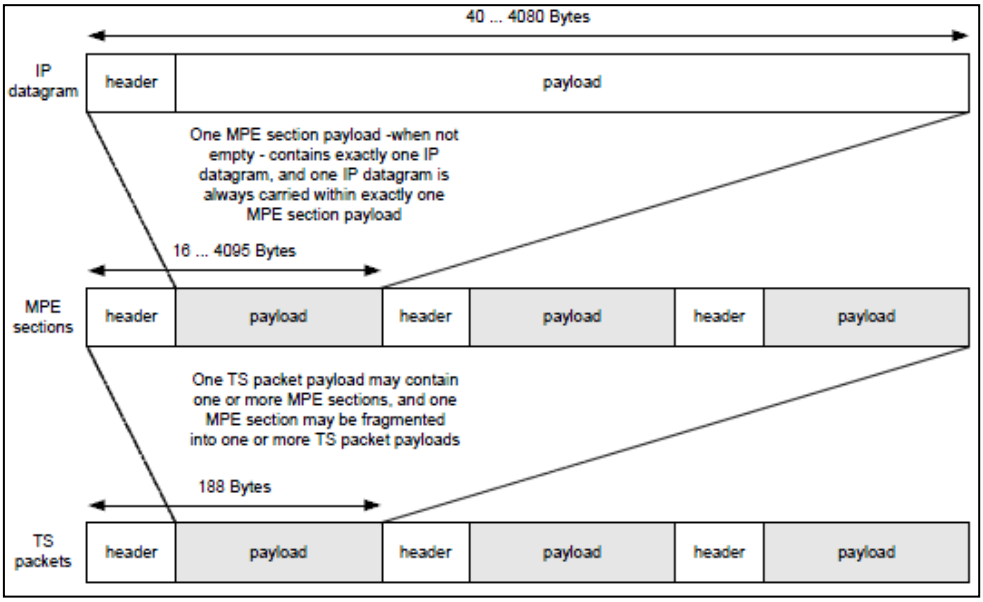


Figura 40. Relación entre el Paquete de Transporte TS, la Sección MPE y el Datagrama IP

La sección MPE para el transporte de un datagrama IP sigue un formato como el que se muestra en la figura 41, en donde el *Section Number* y el *Last Section Number* están configurados a cero por transportar un datagrama IP.

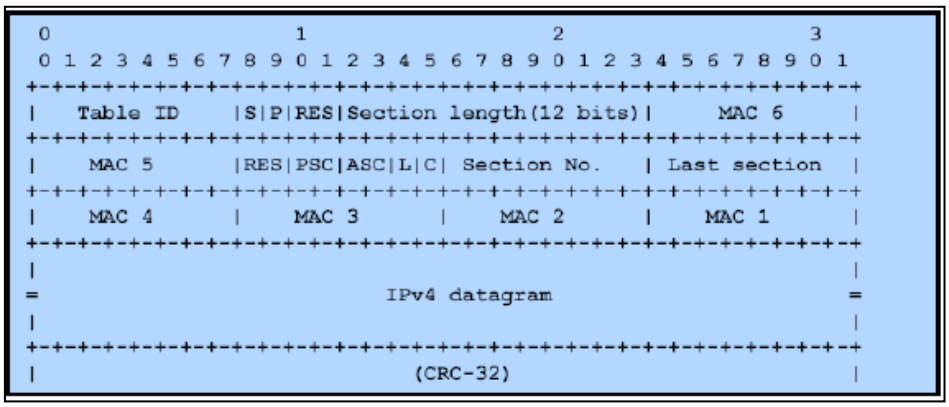


Figura 41.Sintaxis de una Sección MPE-IP

MPE emplea el cambio dinámico de direcciones MAC, figura 42. Esto consiste en dividir los 6 bytes de la dirección MAC en dos grupos dentro de la cabecera MPE. El cambio dinámico de la dirección MAC le permite al receptor filtrar más rápidamente las secciones ya que generalmente los dos primeros bytes de la dirección MAC presentes en la cabecera MPE son filtrados por hardware y son los bytes que probablemente más diferencien un receptor de otro.

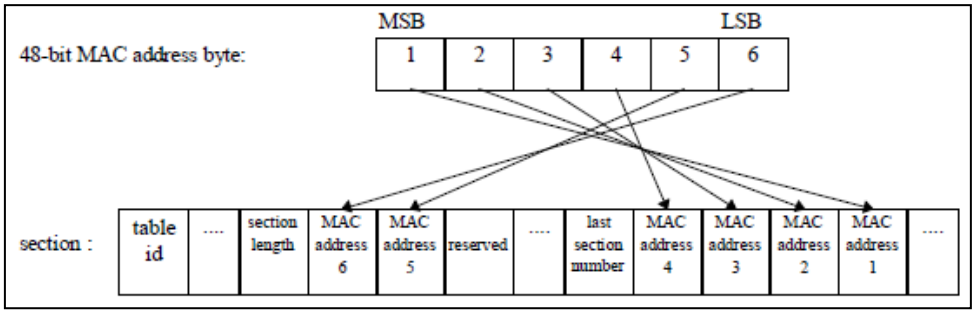


Figura 42. Cambio dinámico de direcciones MAC en MPE

Una característica muy importante del encapsulado MPE es su mecanismo de protección de la señal. Así es que dentro del encabezado tiene un campo denominado *Payload Scrambling control* que indica que la carga útil de la sección esta aleatorizada. Existe también la opción de aleatorizar la dirección MAC a través del campo *Address Scrambling Control*, lo cual provee mayor seguridad gracias al cambio dinámico de la misma.

La sección MPE cuenta con la opción de relleno después del datagrama. Estos bytes de rellenos pueden usarse para hacer que la carga útil de la sección sea múltiplo de un tamaño de bloque determinado, lo cual es normal si se emplea un código de cifrado de bloque. El valor de estos bytes no se especifica y en el caso de que la carga útil esté cifrada se recomienda que el valor de estos bytes no sea fijo, puesto que podría romper el cifrado.

Al final de la sección MPE hay un campo de *checksum* o uno de CRC_32 dependiendo del valor del campo *Section Syntax Indicator*. Se recomienda utilizar CRC_32 puesto que éste provee una mejor protección en contra de los errores de bits, ya que puede ser chequeado por hardware en la mayoría de los demultiplexores (tipo hardware), mientras que el *checksum* se tiene que chequear por software normalmente.

3.3.4. Transporte de paquetes IP dentro del TS MPEG-2

Hay dos formas de transportar paquetes IP dentro del flujo de transporte MPEG-2: una es mediante el encapsulado en un paquete de transporte a través del MPE, y la otra es

el encapsulado en un tren privado, a través del Encapsulado Ligero Unidireccional (ULE).

Como la encapsulación multiprotocolo ya se ha explicado detalladamente en el apartado anterior, ahora se procede a explicar el encapsulado ligero unidireccional.

Encapsulado Ligero Unidireccional (ULE)

El ULE es una técnica para encapsular los datos de los protocolos de las capas superiores en un tren privado dentro del flujo de transporte MPEG-2, está especificado en el IETF RFC 4326.

Un paquete IP transferido de la capa de red se le conoce como Unidad de Datos de Protocolo (PDU). A cada PDU se la encapsula en una Unidad de Datos de Subred (SNDU), añadiendo una cabecera de encapsulación y un verificador. En la siguiente figura se muestra el esquema de una SNDU.

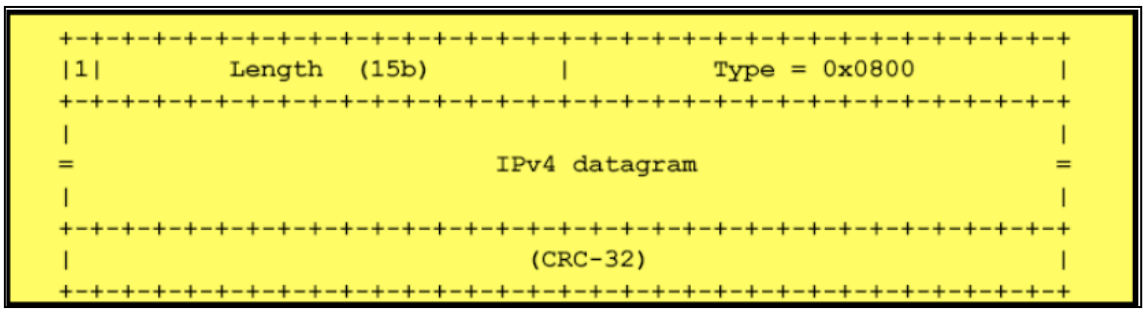


Figura 43. Cabecera mínima ULE

Posteriormente la SNDU es asignada al *payload* de los paquetes de transporte mediante fragmentación. Existen dos procedimientos de asignación: relleno y empaquetamiento. El procedimiento de empaquetamiento es opcional y puede determinarse para casa sesión o para cada SNDU.

En el procedimiento de relleno, figura 44, tras encapsular una SNDU en una serie de paquetes de transporte, no se encapsula inmediatamente otra SNDU aunque haya espacio disponible, no puede haber dos SNDU en un mismo paquete. Este espacio es rellenado por los bits de *padding* hasta completar el paquete.

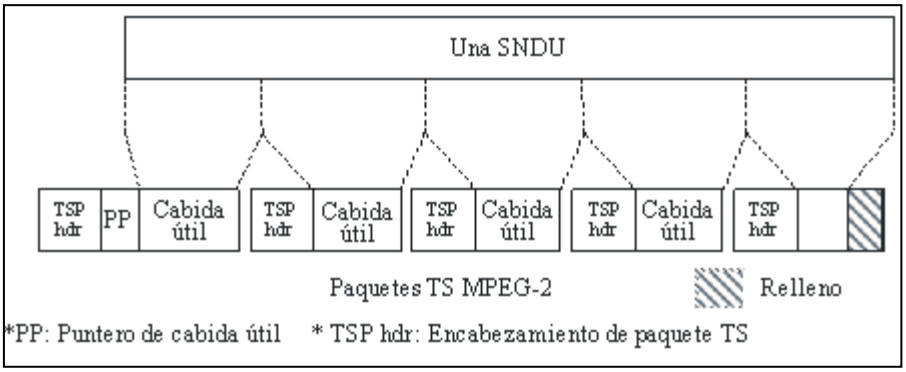


Figura 44. Encapsulado de una SNDU en paquetes de transporte MPEG-2 utilizando el procedimiento de relleno

En el procedimiento de empaquetamiento, figura 45, cuando más de una SNDU está en espera de ser trasferida y a un paquete de transporte le queda espacio suficiente en el *payload*, la SNDU previamente encapsulada va seguida de otra SNDU utilizando el próximo byte disponible de la cabida útil del paquete TS. Para indicar que comienza una nueva sección dentro del paquete de transporte se activa el bit de *Payload Unit Start Indicator* y se añade un nuevo campo al comienzo de la carga útil del paquete que indique cual es la posición exacta en donde empieza la nueva sección, el *Pointer Field*.

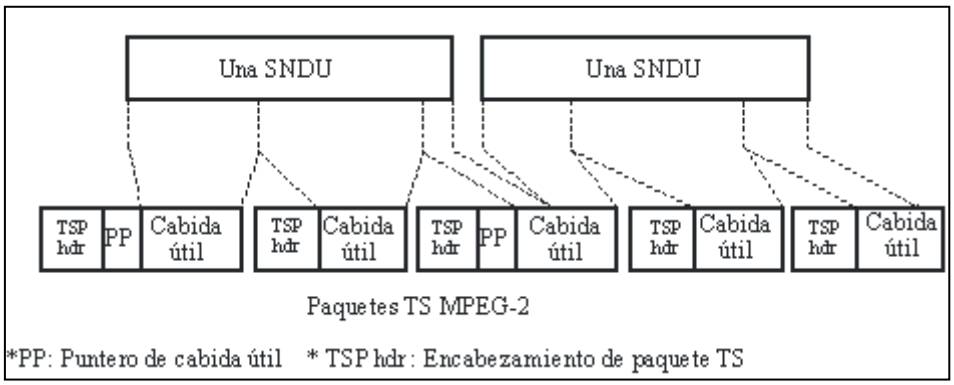


Figura 45. Encapsulado de una SNDU en paquetes de transporte MPEG-2 utilizando el procedimiento de empaquetamiento

3.4. DVB-RCS (*Digital Video Broadcasting – Return Channel Satellite*)

El DVB-S es un estándar pensado para redes unidireccionales en la que un proveedor suministra un contenido a un amplio número de clientes sin esperar una respuesta de

los mismos, es decir, para redes de difusión. En el caso de redes en los que sea necesaria una comunicación bidireccional, que haya una interacción entre los mismos, como es el caso de las redes interactivas VSAT, bastará con incorporar un canal de retorno al sistema. Al principio, para conseguir redes bidireccionales se empleaban canales de retornos terrestres a través de la RTC (Red Telefónica Conmutada) o la RDSI (Red Digital de Datos Integrados). El uso de canales terrestres limitaba la capacidad operativa de estos sistemas ya que se desaprovechaban muchas ventajas inherentes de los sistemas satelitales. Con el fin de crear un canal de retorno vía satélite, el DVB desarrolló el DVB-RCS. Así, para un sistema *full-duplex*, el estándar DVB-S provee los recursos necesarios para el canal de ida (*forward*) de la red VSAT *full-duplex*, y el estándar DVB-RCS especifica el canal de retorno (*return link*) del canal de retorno. Esta funcionalidad se puede ver reflejada en el siguiente esquema.

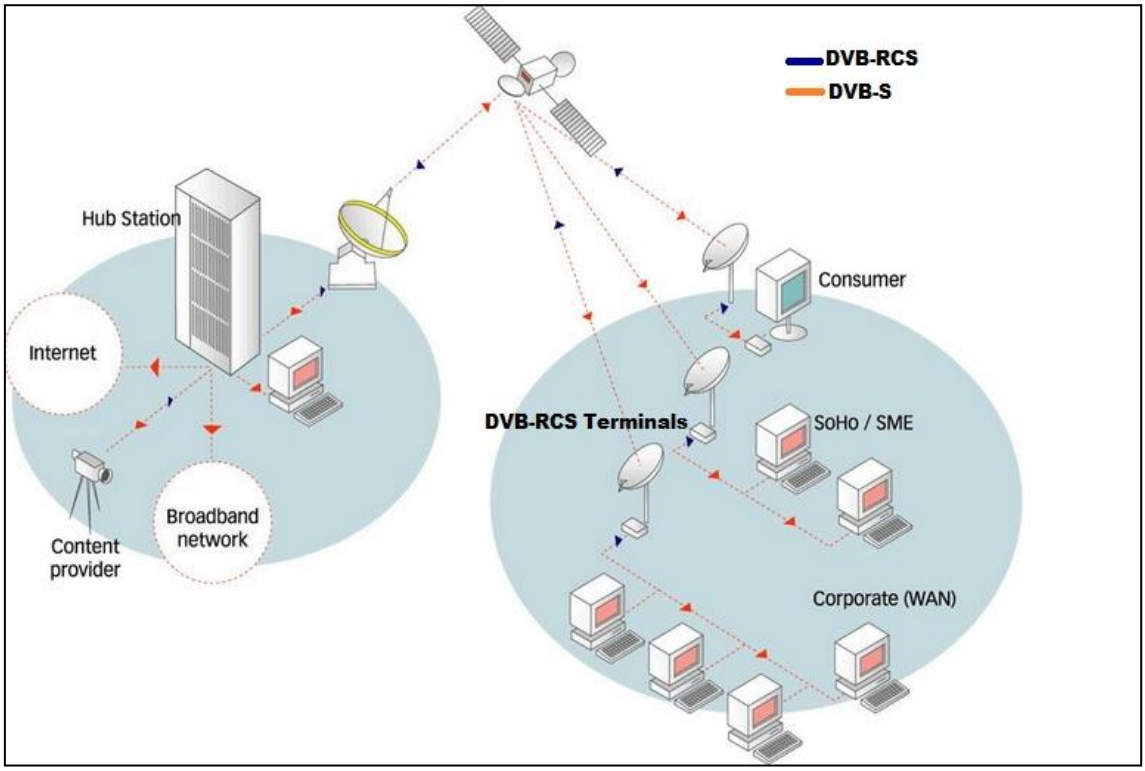


Figura 46. Esquema de enlace DVB-S/RCS

El DVB-RCS fue definido por el DVB y normalizado por la ETSI en 1999 bajo la norma ETSI EN 301 790. Las especificaciones del estándar se desarrollaron en respuesta a las peticiones de varios operadores de red y operadores de satélite que vieron que era esencial tener un estándar abierto que no estuviera ligado a ningún fabricante.

Como se aprecia, todos los estándares del DVB surgen como respuesta a las necesidades del mercado y siguen el mismo principio de poner las bases para el correcto funcionamiento del sistema sin importar el fabricante, es decir, de establecer "las normas del juego".

3.4.1. Modelos de referencia para redes interactivas por satélite DVB

Para estudiar el DVB-RCS existes tres modelos de referencia que se pasan a describir en los siguientes apartados.

3.4.1.1. Modelo de pila de protocolos

Un modelo simple de la pila de protocolos empleada en redes de difusión de servicios interactivos con un canal de retorno satelital consta de las siguientes capas o niveles:

- **Capa física:** donde se definen todos los parámetros físicos para la transmisión.
- **Capa transporte:** define todas las estructuras en la que los datos son empaquetados, así como los protocolos de comunicación.
- **Capa aplicación:** es el software de aplicación que interactúa con el usuario.

La norma DVB-RCS adopto un modelo simplificado del modelo OSI con el fin de facilitar la elaboración de especificaciones para estas capas. Así la figura 47 muestra las capas más bajas de este modelo identificando los parámetros más importantes.

La norma solo define los aspectos que tiene que ver con la Red Satelital Interactiva, es decir, abarca exclusivamente la zona limitada en la imagen como *Network Dependent Protocols*.

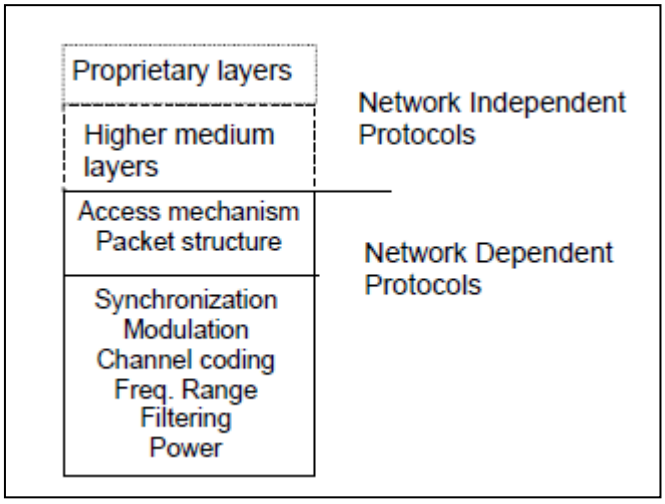


Figura 47. Estructura en capas del modelo simplificado de referencia para DVB-S/RCS

3.4.1.2. Modelo del sistema

Un modelo del sistema es una representación funcional de cómo opera el mismo. En la figura 48 se muestra el empleado por DVB para ofrecer servicios interactivos a los RCST (*Return Channel Satellite Terminal*).

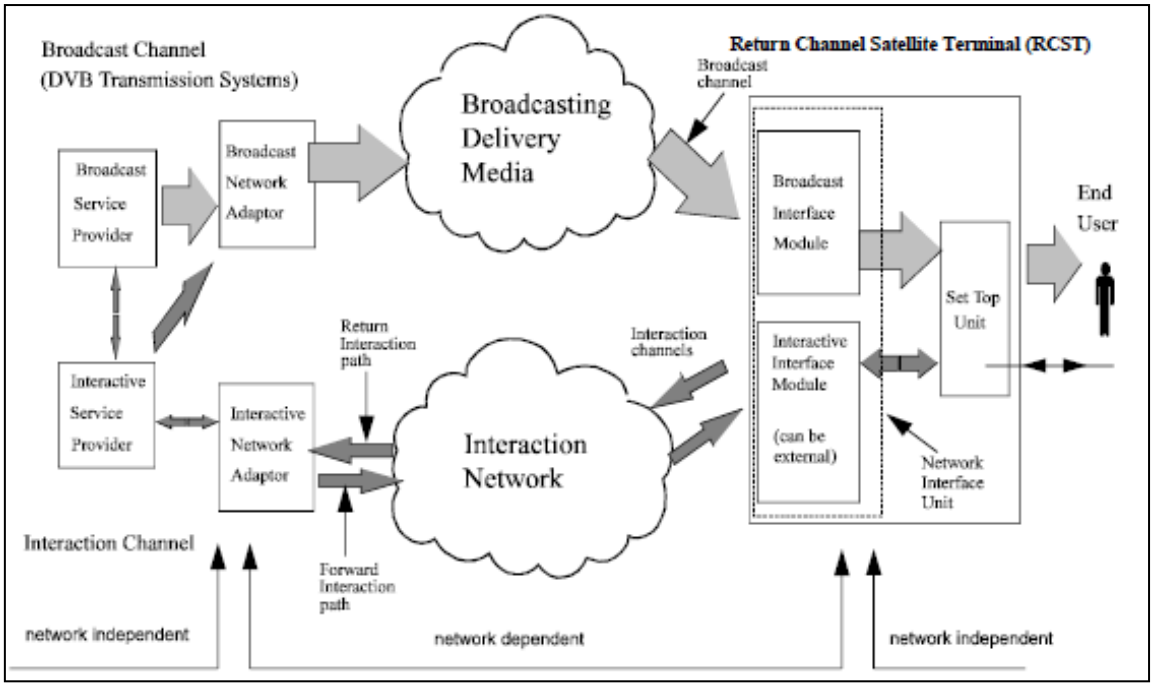


Figura 48. Modelo de referencia de un sistema genérico para sistemas interactivos

El sistema está formado por dos canales de comunicación que se establecen entre el proveedor de servicios y el usuario:

- *Broadcast Channel* (Canal de Difusión): Es un canal unidireccional definido desde el proveedor de servicios al usuario en donde se transporta audio, video y datos de forma multiplexada en un solo flujo (*Transport Stream*). Este puede incluir el *Forward Interaction Path* (Camino de ida de interacción).

- *Interaction Channel* (Canal de Interacción): Es un canal bidireccional establecido entre el proveedor de servicios y el usuario, compuesto por:

    - *Return Interaction Path* (Canal de retorno): Dirigido en el sentido del usuario al proveedor de servicios o a otros usuarios. Se emplea para transferir datos del usuario al proveedor de servicios (realizar peticiones, responder preguntas, etc.) o para interactuar con otros usuarios de la red, es el canal para transferir los datos procedentes del RCST.

    - *Forward Interaction Path* (Canal de Interacción): Es el canal que se establece desde el proveedor de servicios a un usuario. Se utiliza para proporcionar los servicios del proveedor a los usuarios, así como para cualquier otra comunicación necesaria para la provisión de servicios interactivos. Puede ser parte del *Broadcast Channel*. Es posible que este canal no esté como tal en algunas implementaciones de sistemas interactivos.

El RCST está formado por la *Network Interface Unit* (Unidad de Interfaz de Red) que consta de dos módulos, el Módulo de Interfaz de Difusión y del Módulo de Interfaz Interactivo, y la *Set Top Unit* (Unidad Principal de Configuración). El RCST provee el interfaz para ambos canales, el difusivo y el interactivo.

3.4.1.3. Modelo de referencia para redes interactivas por satélite

Como en cualquier red en la que hay un gran número de estaciones trabajando es necesaria la existencia de bloques funcionales que controlen el comportamiento del sistema. Para el caso concreto que se está estudiando el esquema queda de la siguiente manera.

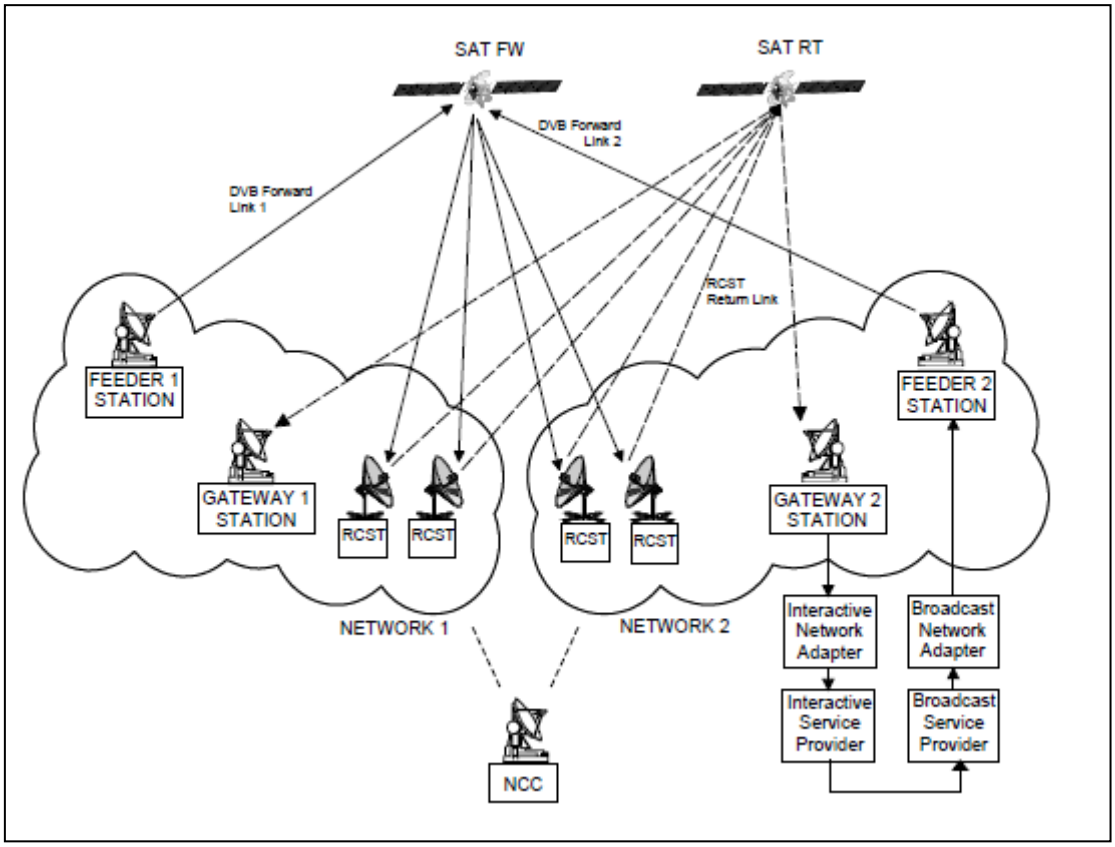


Figura 49. Modelo de referencia para redes interactivas por satélite

Dentro de la red hay tres bloques imprescindibles, el NCC, el TG y el *Feeder*, todos ellos localizados dentro del hub.

- ***Network Control Centre (NCC)*:** El NCC, como su propio nombre indica, es el centro de control de la red. Se encarga de las funciones de control y monitorización, de la administración de la base de datos y de la asignación de recursos para asegurar el correcto funcionamiento de toda la red. Las señales generadas por el NCC son transmitidas a la red a través de una o varias estaciones *Feeder*.

- ***Traffic Gateway (TG)*:** Un TG o puerta de enlace recibe las señales procedentes de los RCST, permitiendo funciones de interconexión con todo tipo de redes y proveedores terrestres (redes datos, Internet, RDSI, base de datos, etc.).

- ***Feeder*:** El *Feeder* o alimentador entrega las señales generadas por el NCC a los RCST de acuerdo con la norma DVB-S, creando el enlace de ida *Foward Link*.

La función del enlace directo de difusión DVB-S es la de transportar las señales de señalización generadas por el NCC, a través del *Forward Link Signaling* (FLS), y los datos generados por el usuario.

3.4.2. Enlace directo (*Forward link*)

El enlace directo proporciona un servicio punto a multipunto, la señal se envía desde una estación en un punto particular a muchas estaciones en puntos diferentes. Sigue la norma DVB-S descrita en el apartado 3.3 y tiene una única portadora que puede emplear todo el ancho de banda del transpondedor (limitado por el ancho de banda) o usar toda la potencia disponible en el mismo (limitado en potencia). Las estaciones comparten el canal usando diferentes *slots* temporales, TDM (*Time Division Multiple*).

3.4.3. Canal de retorno (*Return link*)

Los terminales RCST comparten la capacidad del canal de retorno de uno o varios transpondedores del satélite para transmitir en ráfaga, empleando MF-TDMA (*Multi-Frequency, Time Division Multiple Access*). En un sistema, esto significa que hay un conjunto de frecuencias portadoras para el canal de retorno, cada una de las cuales están divididas en secciones temporales o *time slots* que son asignadas a los terminales por el NCC permitiendo que muchos terminales puedan transmitir simultáneamente al hub.

3.4.3.1. Esquema de transmisión de un RCST

La figura 50 es un diagrama de bloques que representa el proceso por el que pasa una señal en el RCST para ser transmitida a la red. Se pueden apreciar cinco bloques funcionales que forman parte de este proceso:

- Sincronización del RCST.
- Formato de Ráfaga.
- Dispersión de Energía.

- Codificación de Canal.
- Modulación.

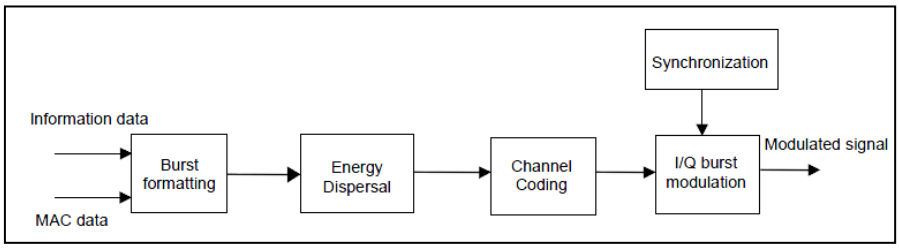


Figura 50. Diagrama de bloques del procesamiento de la señal en banda base del canal de retorno del RCST

Cada uno de los bloques funcionales mencionados anteriormente se describe en los próximos apartados.

3.4.3.2. Sincronización del RCST

Control de Temporización

Una de las características más importantes dentro del sistema es la sincronización del terminal con la red satelital. De hecho, es el primer paso antes de poder transmitir. Existen severas restricciones a los RCST con el objetivo de obtener un sistema TDMA eficiente que consiga mantener mínimas interferencias y máximo rendimiento en la transmisión entre los cientos de terminales activos al mismo tiempo en la red. Cualquier desajuste en la sincronización entre el usuario y el hub provoca que el terminal se desenganche de la red y tenga que volver a engancharse al sistema, ajustando los parámetros de sincronismo.

El esquema de sincronización del sistema se basa en la información contenida dentro del FLS, esta información es:

- El NCR (Reloj de Referencia de la Red)
- Señalización de tiempo dentro de las secciones privadas de DVB/MPEG2-TS

El NRC es distribuido dentro del flujo de transporte MPEG-2 con un PID específico. La distribución del NRC sigue el mismo mecanismo de distribución que el PCR, el cual normalmente se deriva de un codificador de video MPEG, mientras que el NCR

proviene del reloj de referencia del NCC. Este reloj tiene que tener una precisión de al menos 5 ppm.

Sincronización de la portadora

El flujo de transporte MPEG2 que transporta el *Forward Link Signalling* contiene la información del NCR, el cual provee una referencia de 27 MHz a los RCST, permitiéndoles de esta manera alcanzar una sincronización con respecto a la portadora.

Sincronización de la ráfaga

Los RCSTs obtienen la frecuencia central, el tiempo de inicio y la duración de la ráfaga mediante el análisis del FLS. Las ráfagas son enviadas siguiendo un Plan de Asignación de Ráfagas o BTP (*Burst Time Plan*) organizado por el NCC y enviado a través del FLS.

Sincronización del Reloj de Símbolo

La precisión del reloj de símbolo debe de estar dentro de las 20 ppm del valor nominal de la velocidad de símbolo. La velocidad del reloj de símbolo tendrá una estabilidad a corto plazo que limitará el error de tiempo de cualquier símbolo dentro de una ráfaga a 1/20 de la duración del símbolo

3.4.3.3. Formato de ráfaga

El estándar DVB-RCS define cuatro tipos de ráfagas, cada una con una función específica: Ráfagas de Tráfico (TFR), Ráfagas de Adquisición (ACQ), Ráfagas de Sincronización (SYNC) y Ráfagas de Señalización de Canal Común (CSC). Todas las ráfagas vienen precedidas de un preámbulo de tamaño variable para su detección.

Ráfagas de Tráfico (TFR)

Las Ráfagas de Tráfico son las encargadas de transportar los datos útiles desde el RCST hasta el hub. Existen dos formatos de ráfagas de tráfico, que son:

- Ráfagas ATM: El formato ATM está permitido como opción para el canal de retorno. La carga útil está formada por paquetes ATM concatenados de 53 bytes de longitud y un byte opcional de prefijo. La siguiente figura muestra la estructura de este tipo de ráfaga.

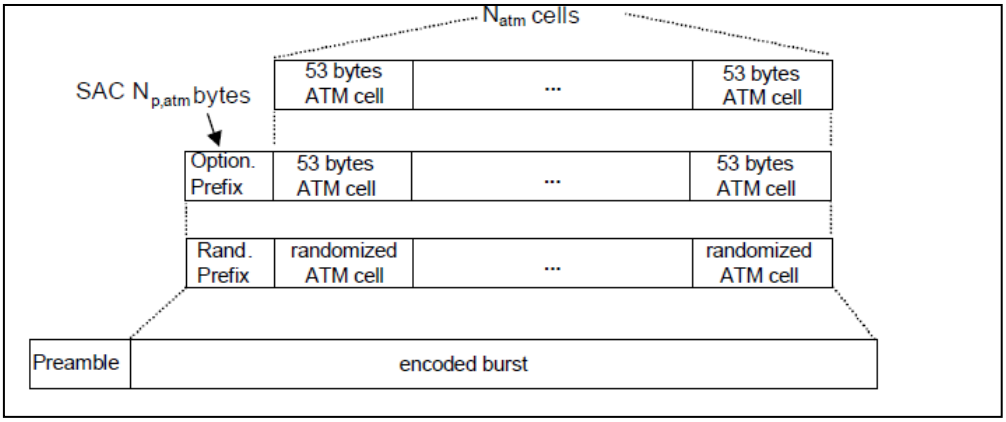


Figura 51. Composición de una ráfaga de tráfico ATM

- Ráfagas MPEG-2 TS: La carga útil está compuesta por una sucesión de paquetes MPEG-2 TS concatenados de 188 bytes cada uno. La figura 52 muestra la estructura de una ráfaga de tráfico tipo MPEG2.

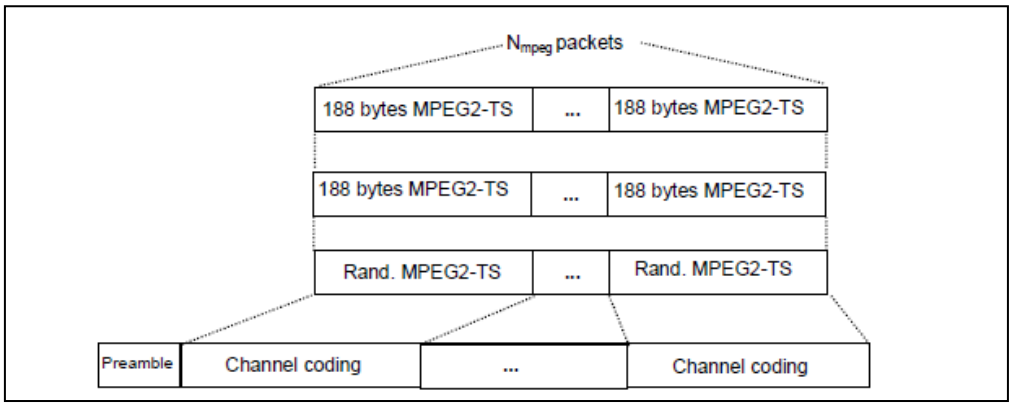


Figura 52. Composición de una ráfaga de tráfico MPEG2

El envío de una TFR viene seguido por un tiempo de guarda para permitir al RCST disminuir su potencia de transmisión y así no interferir con el envío de las siguientes TFR.

Ráfagas de Sincronización (SYNC) y Adquisición (ACQ)

Las Ráfagas de Sincronización SYNC y Adquisición ACQ son requeridas para indicar la posición exacta de las ráfagas a ser transmitidas por un RCST durante y después del proceso de *logon* (proceso de entrada al sistema). Se emplean para la sincronización de un RCST.

- Ráfaga de Sincronización SYNC: Las ráfagas SYNC se utilizan para mantener la sincronización de los RCST, así como la de enviar información de control al sistema. La ráfaga está compuesta de un preámbulo, configurable e indicado al terminal a través de la tabla TCT (*Time-slop Composition Table*), y un campo opcional de un byte denominado *SAC_length* que forma el SAC (*Satellite Access Control*). Al igual que una TRF, hay un intervalo de guarda después de una SYNC. La norma ha dejado abierto el tamaño de la ráfaga SYNC y dependerá exclusivamente de las capacidades del NCC. La figura 53 muestra dicho formato.

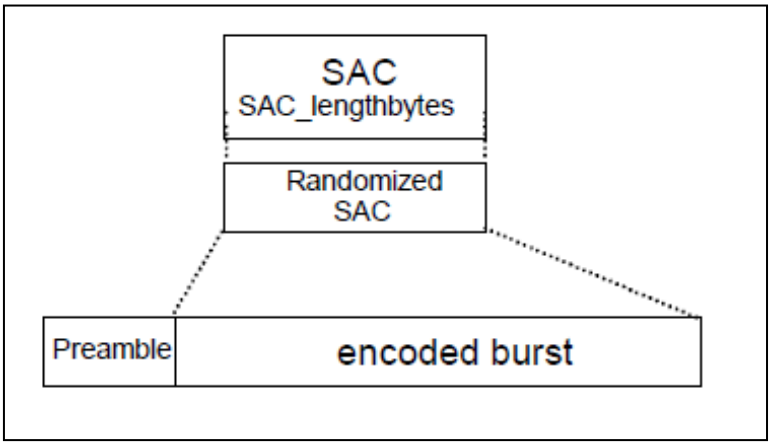


Figura 53. Composición de una ráfaga SYNC

- Ráfaga de Adquisición ACQ: Un terminal emplea la ráfaga ACQ para lograr sincronizarse con el sistema antes de su puesta en funcionamiento. El formato de esta ráfaga se muestra en la siguiente figura, en donde se puede ver que está compuesta por un preámbulo seguido de una secuencia de frecuencia, ambos obtenidos de la tabla TCT.

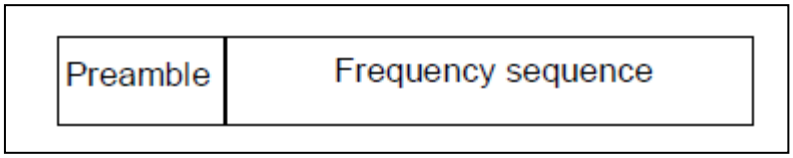


Figura 54. Composición de una ráfaga ACQ

Ráfagas de Señalización de Canal Común (CSC)

Las Ráfagas CSC de acceso aleatorio son únicamente usadas por el RCST para identificarse durante el proceso de carga en la red.

La unidad RCST envía ráfagas CSC al hub para una petición de acceso a la red. El hub comprueba que la unidad RCST contiene una serie de parámetros necesarios para su entrada en la red, tales como su capacidad y la dirección MAC. Una vez comprobado esto, el hub le envía una ráfaga de respuesta para permitirle el acceso a la red. La figura 55 representa la estructura de una ráfaga CSC.

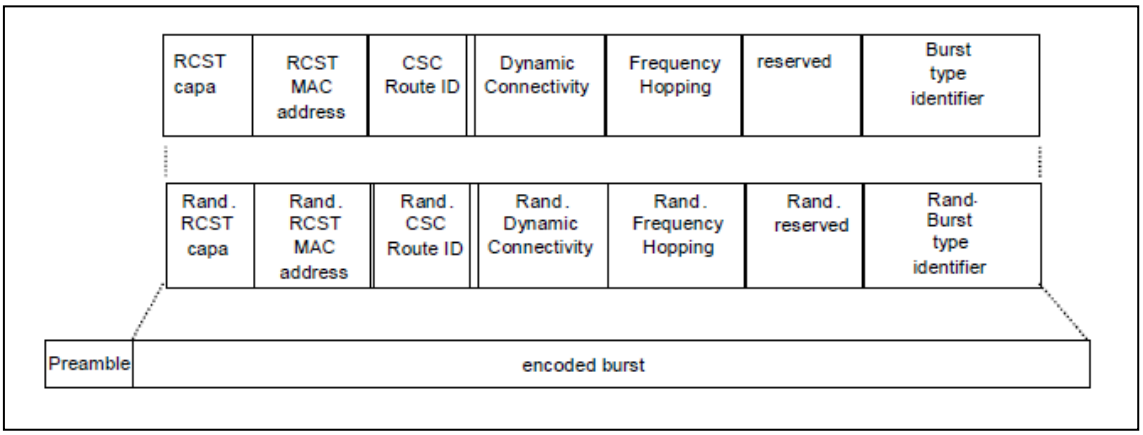


Figura 55. Composición de una ráfaga CSC

3.4.3.4. Aleatorización para dispersar la energía

Los datos del canal de retorno deben ser organizados en ráfagas tal y como se ha descrito en el último apartado. Ahora, con el objetivo de cumplir la normativa y hacer una transmisión binaria adecuada, el flujo serial de bits de DVB-RCS debe ser aleatorizado. Este proceso es el similar al llevado a cabo por DVB-S, explicado anteriormente.

3.4.3.5. Codificación

La codificación para protección de errores se aplica tanto al tráfico de datos como al de control. Para la codificación se tienen dos esquemas: Codificación Turbo y Concatenada. Los terminales RCST tienen que ser capaces de implementar ambos esquemas, pero solo emplearan una por cada sesión, no ambas. En el caso de la codificación concatenada, la codificación externa es *Reed-Solomon* y la interna es un bloque convolucional (al igual que en DVB-S). Para ambos esquemas de codificación se puede emplear un código CRC para la detección de errores en las ráfagas CSC y SYNC. Si esto se va a emplear el NCC se lo informa a los RCST mediante la tabla TCT.

3.4.3.6. Modulación

La señal a ser transmitida por el canal de retorno, una vez ha sido codificada, debe seguir un esquema de QPSK, al igual que DVB-S. Este proceso es el que se recoge en la siguiente figura.

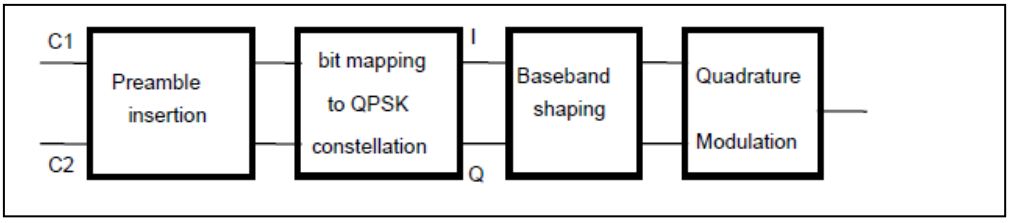


Figura 56. Procesado de la señal después del codificador en DVB-RCS

El preámbulo se configura e indica a los RCST a través de la tabla TCT. Inmediatamente después de la inserción del preámbulo al flujo de bits, la salida C1 y C2 del codificador se envían sin modificación al Mapeador de Bits o *Bit Mapper QPSK*.

Se usa una modulación QPSK convencional en combinación con un código Gray. Se aplica un factor de normalización ½ a las componentes I y Q, dando lugar a que la energía media por símbolo sea igual a 1. La constelación resultante queda reflejada en la figura 57.

La salida C1 del canal de codificación se asigna al canal I de la modulación y la salida C2 al canal Q.

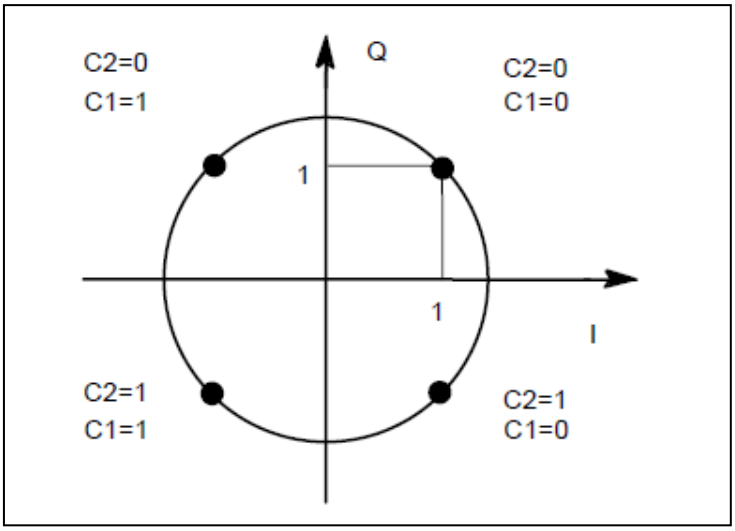


Figura 57. Diagrama de constelación para QPSK con código de Gray

3.4.3.7. Mensajes MAC

Son los mensajes empleados por los RCST para controlar el acceso al medio de los mismos. Estos son los encargados de realizar las peticiones de capacidad y del envío de mensajes de control y administración.

La norma específica varios métodos para el envío de mensajes MAC, pero normalmente suelen ser divididos en solo dos métodos: los basados en el Campo SAC y los basados en el envío de Unidades DULM (*Data Unit Labelling Method*) dentro del tráfico de usuario.

Métodos basados en el Campo SAC

Este método se basa en la activación del campo SAC presente en las ráfagas SYNC y en el prefijo opcional de las ráfagas TRF ATM y MPEG con el fin de que sean estas las que transporten las peticiones de capacidad y los mensajes MAC dentro de la red.

Método basado en el envío de Unidades DULM

Es un método que se basa en el envío de mensajes dentro de las ráfagas de TFR. Los mensajes se transportan dentro de la carga útil de la Unidad de Datos, que puede ser una celda ATM o un paquete de transporte MPEG, dependiendo del tipo de encapsulamiento escogido en el RCST.

La secuencia de estas ráfagas TFR especiales forman un canal virtual dedicado conocido como CTRL/MNGM (*Control/Management*) por donde se transmiten mensajes de control e información de administración, posiblemente en combinación con otros métodos empleados para el envío de mensajes MAC.

3.4.4. Acceso al medio de la red DVB-RCS

En una red interactiva DVB-S/RCS el acceso al medio por parte de los terminales RCST se logra mediante un esquema MF-TDMA. Este esquema permite a un grupo RCST comunicarse con la *Gateway* usando un conjunto de frecuencias portadoras, cada una de las cuales divididas en espacios temporales (*time-slots*). El NCC asigna a cada RCST una serie de ráfagas para transmitir, cada una de las cuales definidas por una frecuencia centra, un ancho de banda, un tiempo de inicio y una duración de la misma.

La asignación de las capacidades del satélite a los RCST pueden ser fijas o dinámicas. El RCST será el encargado de informar al NCC sobre su capacidad empleando el campo MF-TDMA presente en la ráfaga CSC.

MF-TDMA Fija

En el esquema de *slots fijos* el ancho de banda y duración de los slots empleados por un RCST son fijos, tal y como muestra la figura 58 en donde la flecha indica la secuencia de slots sucesivos asignados por el NCC a un RCST. La tabla TCT es la encargada de hacer llegar al terminal todos los parámetros de la ráfaga.

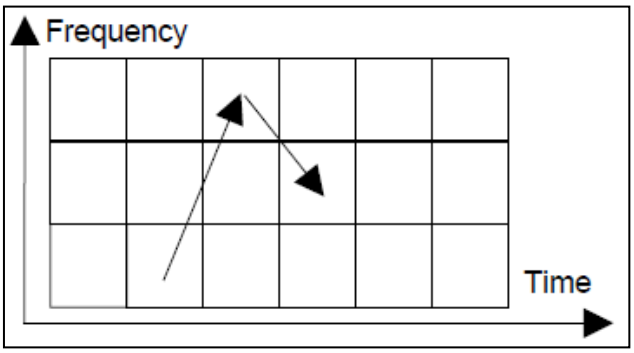


Figura 58. MF-TDMA con slots fijos

MF-TDMA Dinámica

En este esquema los slots sucesivos asignados a un RCST pueden variar su ancho de banda y la duración de los mismos utilizando la flexibilidad de los RCST, tal y como se muestra en la figura 59. También es posible cambiar la velocidad de transmisión y de codificación entre ráfagas consecutivas. Todos estos cambios permiten una transmisión más eficiente que se adapte a las necesidades cambiantes del medio.

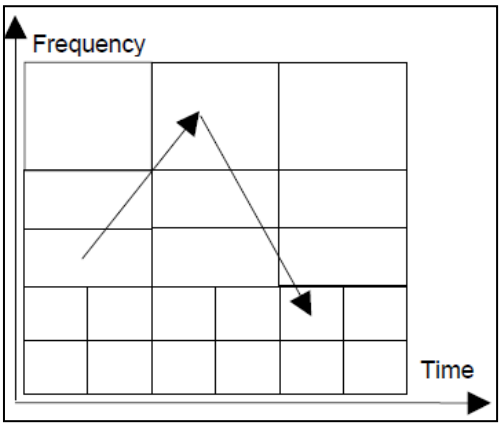


Figura 59. MF-TDMA con slots dinámicos

4. SEGURIDAD EN LA RED VSAT DVB-S

La privacidad en las comunicaciones deber estar asegurada evitando la intromisión de cualquier usuario no autorizado. Especialmente importante es en las redes por satélite en donde cualquier dispositivo que se encuentre dentro del haz del satélite podrá recibir dicha señal. Así, la seguridad debe de garantizar:

1. **Autentificación**. Asegurar que el dispositivo con el que se establece la conexión es realmente con el que se quiere y no otro.
2. **Confidencialidad**. Evitar que cualquier terminal no autorizado que este interceptando la comunicación pueda extraer la información.
3. **Integridad**. Garantizar que los datos recibidos por el receptor no han sido alterados por un tercero y son los que realmente envió el transmisor.

Para llevar esto acabo existen varias niveles de seguridad que van desde la seguridad propia de DVB-S/RCS a nivel de enlace, hasta los protocolos de seguridad en las capas más altas. Se puede ver un esquema de lo mencionado en la figura 60.

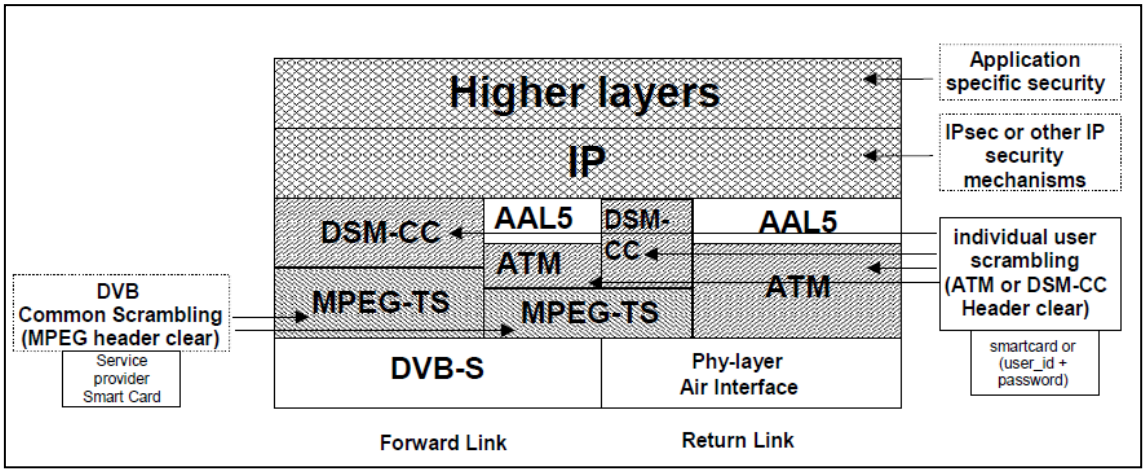


Figura 60. Capas de seguridad en una red interactiva DVB-S/RCS

4.1. Seguridad *Common Scrambling Algorithm*

El *Common Scrambling Algorithm* (CSA) es el método empleado por los sistemas DVB para proporcionar un acceso condicional a los servicios mediante la encriptación del flujo de transporte MPEG-2. Fue adoptado en 1994, aunque la fecha exacta del diseño es incierta ya que se mantuvo en secreto varios años por cuestión de seguridad.

Los detalles del CSA no han sido publicados ya que están ligados con la seguridad de la difusión de las señales, aunque las compañías pueden acceder a esas especificaciones bajo la firma de un contrato de confidencialidad. El hecho de tener un algoritmo de seguridad secreto no transmite mucha confianza ya que no puede ser estudiado por la comunidad en busca de posibles puntos débiles. Esto contrasta con la filosofía de los nuevos sistemas de seguridad basados en algoritmos abiertos en donde expertos en seguridad ajenos a la norma pueden estudiarlo y descubrir sus puntos débiles con objeto de corregirlos.

CSA está formado, como se muestra en la figura 61, por un conjunto de dos bloques criptográficos puestos en cascada: en primer lugar se realiza un cifrado en bloque (*block cipher*) de 64 bits y a continuación un cifrado de bit a bit (*stream cipher*). Ambos emplean la misma llave (*Control Word*) para encriptar y desencriptar los datos, cifrado simétrico, por lo que con atacar a uno de los bloques se podría romper el algoritmo.

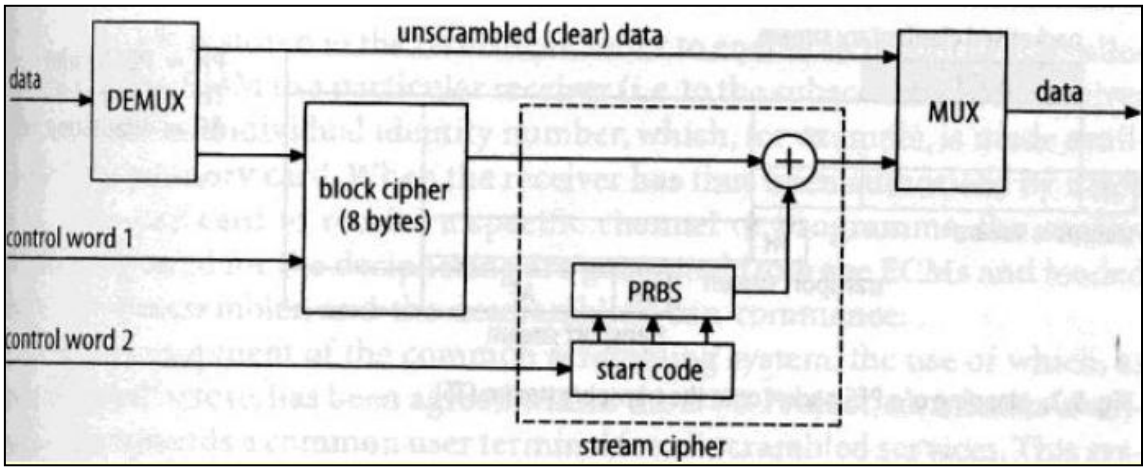


Figura 61. Bloque de cifrado CSA

Independientemente del proceso de cifrado escogido por cada proveedor, los sistemas de acceso condicional DVB siguen una estructura definida por la norma en donde hay tres piezas clave:

- *Control Word:* clave secreta de 48 bits generada aleatoriamente en el NCC con el objetivo de cifrar los datos en claro de acuerdo con el algoritmo CSA.
- *Service Key*: clave propia para cada servicio encriptado.
- *User Key (Cookie)*: clave única de cada terminal que lo diferencia del resto; solo la conocen el NCC y el propio terminal.

La información a mandar se cifra con la *Control Word.* La *Control* Word se cifra con la *Service Key*, proporcionando un primer nivel de cifrado, y la *Service Key* se cifra con la *User Key*.

Cada servicio tiene una *Service Key* diferente que permite cifrar los servicios individualmente, dando permisos a unos terminales y a otros no. Esta *Service Key* será común para todas las estaciones que tengan contratado dicho servicio. Además, como cada terminal tiene una *User Key* diferente, es necesario cifrar la *Service Key* con cada una de las *User* Key que tenga permiso para acceder a ese servicio. En el decodificador de cada terminal se analizará si la *Service Key* viene cifrada con su *User Key*, y si es así, se procederá a su descodificación.

La desencriptación se hace antes de llegar al decodificador MPEG en un hardware externo usando la información extraída de los flujos ECMs y EMMs, junto con el *User Key* almacenado en la *Smart Card* del terminal. Este proceso se muestra en la figura 62.

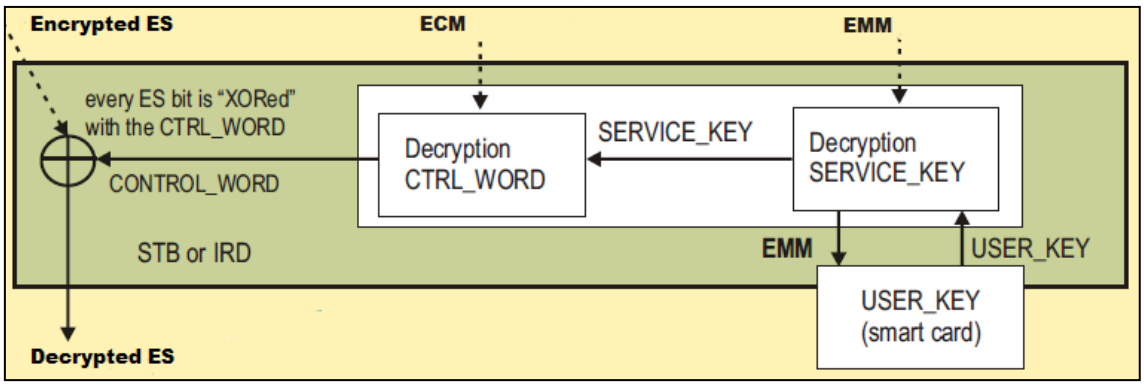


Figura 62. Esquema *Descrambling* DVB-S.

El CSA no proporciona un servicio de seguridad por terminal, sino por servicio, emplea una *Control Word* única por servicio que es enviada a los terminales a través del flujo ECM. Esto está bien para la difusión de contenidos vía satélite pero no para el intercambio de datos entre dos estaciones concretas, ya que los datos podrían ser desencriptados por cualquier otra estación. Además no proporciona ningún mecanismo de seguridad para el canal de retorno en la red DVB-S/RCS.

4.2. Seguridad *Individual Scrambling*

El modo *Individual* Scrambling es una variación del *Common Scrambling Algorithm* citado anteriormente. Aquí en vez de existir una única *Control* Word por programa se estable una *Control* Word diferente entre cada estación RCST y el NCC. Esto se consigue gracias al *User* Key que tiene cada RCST, que permite identificarla dentro de la red.

Obtención de la *Control Word*

Cuando se inicia una nueva sesión entre el terminal y el NCC, antes de transferir ningún dato, se produce un intercambio de mensajes MAC con el fin de establecer una clave compartida que solamente ellos dos conocen y que se utilizará para encriptar los datos, conocida como la *Control Word* o *Session Key.*

El primer paso para alcanzar la *Control Word* consiste en la autentificación del RCST. Para ello el NCC puede emplear un software que mediante la introducción de un *username* y una *password* compruebe si está en la base de datos de dispositivos permitidos. Otra forma de autentificar al usuario es mediante una *Smart Card* alojada en el terminal RCST que contiene una clave secreta, el *User Key*¸ y que solamente conocen el NCC y el terminal.

Una vez asegurada la autenticidad del RCST que quiere conectarse a la red, comienza un intercambio de mensajes entre el NCC y el RCST para definir un perfil de seguridad (algoritmo criptográfico a emplear y tamaño de las claves). Esto se realiza mediante los mensajes de *Security Sign-On* y *Security Sign*-On *Response*, figura 63.

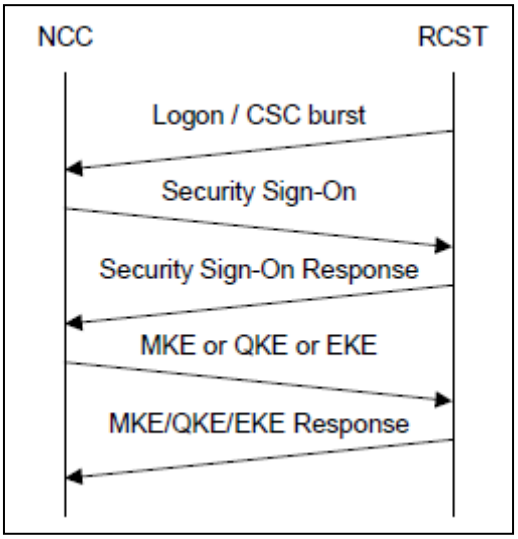


Figura 63. Establecimiento de la seguridad en DVB-RCS

Hay tres mecanismos para la autentificación e intercambio de claves:

- *Main Key Exchange (MKE)*: emplea el protocolo criptográfico *Diffie-Hellman* para desarrollar un secreto compartido temporal (*Share Secret*) a través de un medio inseguro. La autentificación la consigue por medio del *Cookie* del RCST; opcionalmente se puede emplear el *Share Secret* para realizar una actualización de la *Cookie.* Finalmente se obtiene la *Control Word* a través de una función *hash* que tiene como parámetro el *Share Secret* anterior.
- *Quick Key Exchange (QKE)*: emplea el *Cookie* para autentificar el RCST en el NCC y obtener un *Share Secret*. Al igual que en el mecanismo anterior, la *Control World* se obtiene a través de este *Share Secret*.
- *Explicit Key Exchange (EKE):* el NCC transmite una predeterminada *Control Word* al RCST, esta está encriptada empleando una clave temporal proveniente del valor del *Cookie*.

4.3. Seguridad a nivel de red

El nivel de red por sí mismo no presenta ningún medio de seguridad para proteger los paquetes que circulan por la red. Para solucionar este problema se diseñó el protocolo *IPSec,* abreviatura de *IP Security*. Este protocolo sigue un diseño *end-to-end*, lo que significa que solo el equipo que transmite los datagramas IP y el equipo que los recibe

hacen uso de él, por lo que tanto el equipo transmisor como el receptor tienen el conocimiento del algoritmo de cifrado y descifrado de los paquetes enviados.

El protocolo *IPSec* tiene las siguientes características en su funcionamiento:

- Garantiza la integridad de los paquetes que se transmiten por la red. Cuando el equipo final recibe los datagramas, si la suma de verificación es incorrecta significa que el datagrama ha sido alterado y por lo tanto no es seguro, por lo que se desecha.
- Autentica que el usuario que recibe los datagramas es el correcto, si no lo es, no puede descifrar el paquete y por lo tanto no puede acceder a la información.
- Permite filtrar por dirección IP, configurando un rango o una IP en concreto. También permite el bloqueo de paquetes por puertos TCP.
- Compatible con el cifrado de los datos de los niveles superiores

Este protocolo está formado por dos protocolos para la autenticación y el cifrado de datos, el protocolo AH (*Authentication Header* - Cabecera de autenticación) y el protocolo ESP (*Encapsulated Security* Payload - Carga de seguridad encapsulada). Cada protocolo es independiente, pudiendo hacer uso de cada uno según las características del medio y la seguridad que el usuario quiera integrar.

Protocolo *Authentication Header*

Este protocolo crea una cabecera antes de la cabecera IP que se envía a los niveles inferiores por el protocolo **IP 51**. Su función es la verificación de la integridad de los datos enviados mediante una comprobación HMAC (K*eyed-hash Message Authentication Code*) generada a partir de una clave secreta. La cabecera AH tiene la siguiente estructura de 24 bytes.

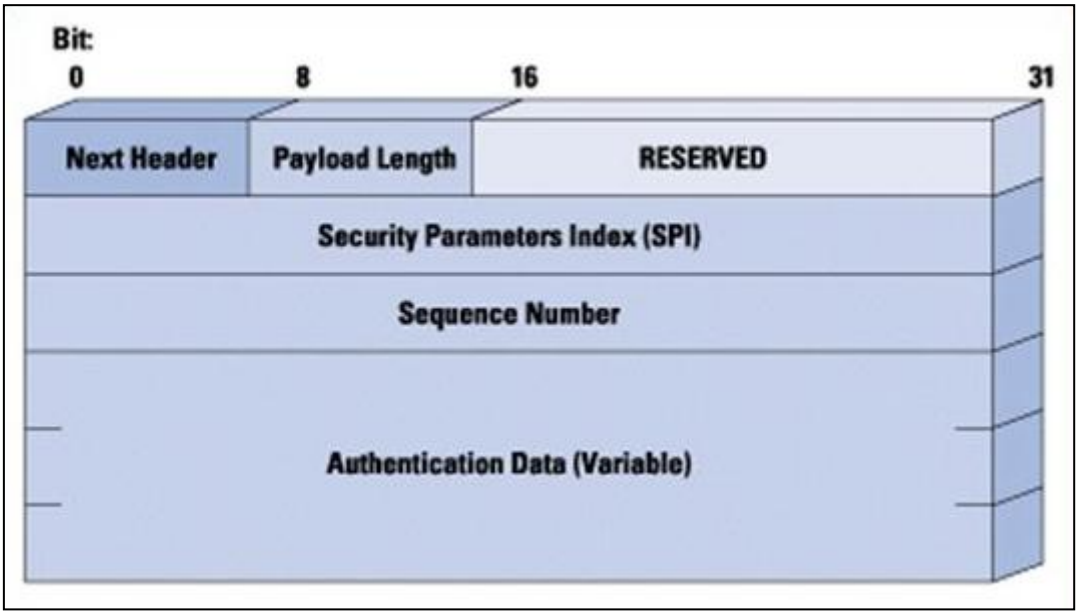


Figura 64. Estructura Protocolo AH

- *Next Header (8 bits).* Indica cual es el protocolo siguiente empleado. Si la transmisión se realiza en modo túnel, toma el valor 4, si es en modo transporte toma el valor 6.
- *Payload Length (8 bits)*. Muestran la longitud de los datos.
- *Reserved (2 bytes)*. Campo reservado.
- *Security Parameter Index* o *SPI (4 bytes)*. Campo que indica mediante la Asociación de Seguridad (SA) la forma de desencapsular el datagrama.
- *Sequence Number (4 bytes)*. Número de Secuencia que protege al paquete contra ataques de repetición.
- *Authentication Data (12 bytes)*. Almacena el valor del código HMAC. Comprueba que la integridad del paquete no ha sido alterada mediante una clave compartida entre el usuario que envía el paquete y el que lo recibe.

Protocolo *Encapsulated Security Payload*

El protocolo además de asegurar la integridad de los datos, también hace el cifrado de estos por el protocolo **IP 50**. La integridad es asegurada mediante la comprobación HMAC, que calcula que no se han modificado los bits del paquete, mientras que la confidencialidad se realiza a través del vector de cifrado, transmitiendo los datos de

forma ilegible hasta el usuario final. La cabecera que genera este protocolo es la siguiente:

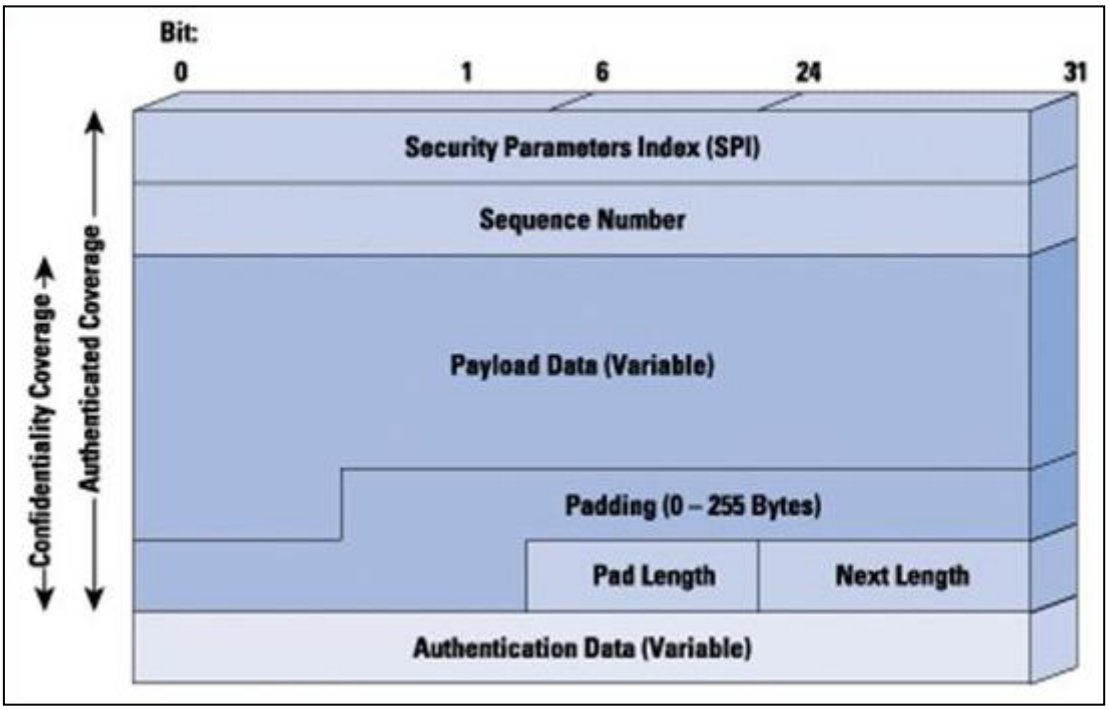


Figura 65. Estructura Protocolo ESP

Este protocolo añade una cabecera y unos bits de cola al final del datagrama del nivel superior.

- *Security Parameter Index (4 bytes)*. Indica la forma de desencapsular el datagrama mediante la Asociación de Seguridad.
- *Sequence Number (4 bytes)*. Protege el datagrama ante ataques por repetición.
- *Payload Data*. Es el *payload* del paquete, siendo estos los datos a proteger.
- *Padding*. Posibles bits de relleno necesarios para poder realizar el proceso de cifrado.
- *Pad length (8 bits)*. Indican cuantos bits se han rellenado junto con la carga.
- *Next Length (8 bits)*. Señala cual es la cabecera siguiente empleada.
- *Authentication Data (4 bytes)*. Almacenan el valor del código HMAC para comprobar que no se ha alterado el paquete. Esta HMAC solo tiene en cuenta la carga del paquete, la cabecera IP no se incluye dentro de su proceso de cálculo.

El protocolo *IPSec* tiene dos modos de transmisión dependiendo sobre el nivel que actué.

Modo Transporte

En este modo, a los datagramas IP se le añaden las cabeceras de los protocolos AH y ESP, uno de ellos o los dos a la vez. La cabecera IP no se encapsula, solo los datos de las capas superiores están aseguradas por un *hash* para que no puedan ser modificadas. Este modo es utilizado para las conexiones entre dos usuarios finales debido a que no se utiliza ningún enrutamiento.

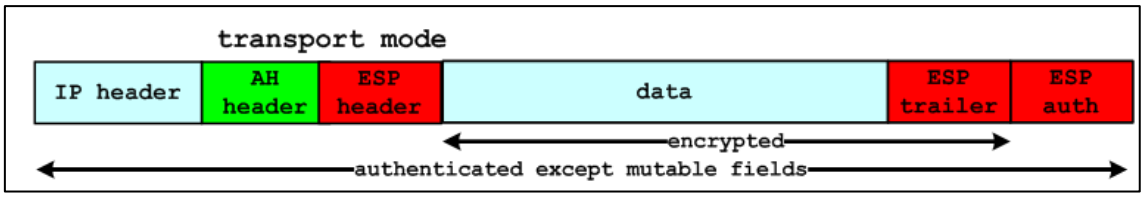


Figura 66. Modo Transporte

Modo Túnel

En el modo túnel, aparte de incluir las cabeceras de los protocolos AH y/o ESP, todo el paquete IP es cifrado generando una nueva cabecera IP para que pueda ser enrutado por la red. Este modo es utilizado para las conexiones entre red y red, red y usuario o usuario a usuario.

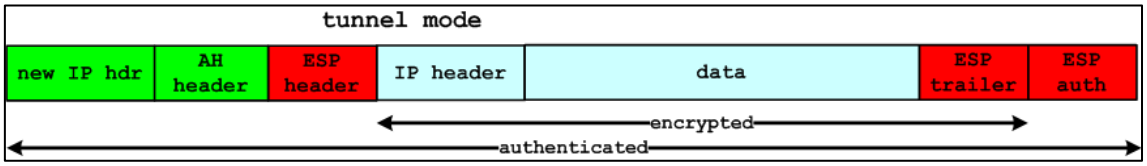


Figura 67. Modo Túnel

4.4. Seguridad en las capas superiores

En las capas superiores del modelo TCP/IP o del modelo OSI, capa de transporte, capa de sesión, capa de presentación y capa de aplicación, también existen protocolos de seguridad. La forma de asegurar los datos es parecida a la que se ha visto con el protocolo *IPSec*, utilizando técnicas de autenticación y de cifrado. En estas capas

existen los protocolos SSL y SSH, ambos utilizan algoritmos de autentificación, de encriptación y verificación de paquetes para las conexiones seguras. Sus funciones son:

- Autenticación. Cliente y servidor se identifican por medio de claves simétricas, públicas o certificados digitales.
- Integridad. Se comprueba que los datos no han sido alterados.
- Privacidad. Los datos viajan encriptados.

4.4.1. Security Sockets Layer / Transport Security Layer

El protocolo SSL, *Security Sockets Layer*, es un protocolo que hay entre el nivel de transporte y el de aplicación. Fue creado por la compañía **Netscape** en la década de los años 90 para proporcionar conexiones seguras en una red entre cliente y servidor, sobre todo en las conexiones HTTP.

En su primera versión 1.0 el protocolo tenía muchas vulnerabilidades por lo que no se llegó a publicar, siendo la versión 2.0 la primera publicación que hizo **Nestcape** en 1995. La última versión es la 3.0 que ya no se emplea en la actualidad ya que este protocolo ha sido reemplazado por el protocolo TLS (*Transport Layer Security*), siendo una continuación del SSL y que en su primera versión mantiene la compatibilidad con la versión 2.0 del protocolo SSL y las siguientes. El TLS tiene como última versión la 1.2 que fue publicada por el grupo IETF (*Internet Engineering Task Force*) en la publicación del documento RFC 6176 en marzo de 2011, con la importante característica de no dar retrocompatibilidad a la versión SSL 2.0.

Este protocolo se diseñó de manera independiente de los niveles de aplicación y transporte, cualquier aplicación puede hacer uso de él e implementarlo. Su función se basa en cambiar los **sockets**, también conocidos como **comandos**, para reconocer que la conexión a realizar es segura. Para establecer una conexión, primero se negocian las claves a utilizar en la transferencia de datos. En el proceso de negociación entre cliente y servidor se establecen los siguientes parámetros:

- Versión del protocolo TLS más alta soportada por ambos.

- Método de autenticación, por clave o por certificado digital.
- Identificador de la sesión.
- Algoritmo para el cifrado de los datos.

La siguiente imagen muestra el intercambio de comandos que hacen cliente y servidor en el proceso de negociación.

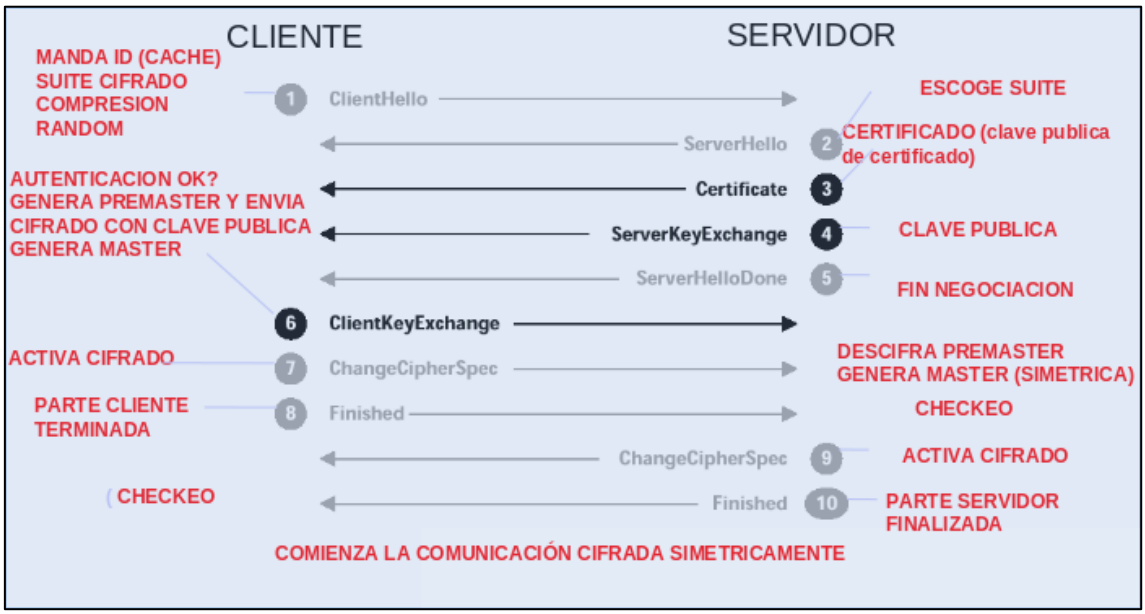


Figura 68. Inicialización Protocolo SSL/TLS

Una vez que se han acordado los algoritmos y la autenticación entre cliente-servidor ha sido correcta, cada paquete es encriptado, verificado, comprimido y enviado según las condiciones de seguridad negociadas. La conexión finaliza cuando ha habido un error grave y se ha alertado al cliente o cuando cliente y servidor mandan un mensaje de fin de conexión.

4.4.1.1. Estructura protocolo SSL/TLS

El protocolo SSL está divido en dos partes, que dependiendo de la fase de conexión. La primera parte está formada por el Nivel de Negociación, formado por el Protocolo de Negociación (*Handshake Protocol*), el Protocolo de Cambio de Cifrado (*Change Cipher Spec Protocol*) y el Protocolo de Alertas (*Alert Protocol*). El Protocolo de Negociación es el encargado de la autenticación entre cliente y servidor y de acordar los algoritmos y

claves que se utilizarán en la transferencia de datos para hacerlo de forma segura, el Protocolo de Cambio de Cifrado realiza el intercambio de claves de cifrado al inicio de la conexión y en cualquier momento de la conexión establecida, y el Protocolo de Alertas envía un mensaje de alerta con el tipo de alerta.

La segunda parte está formada por el Nivel de Registro que solo tiene un protocolo, se encarga de la manipulación de datos del nivel de aplicación para ser enviados con los parámetros de seguridad negociados.

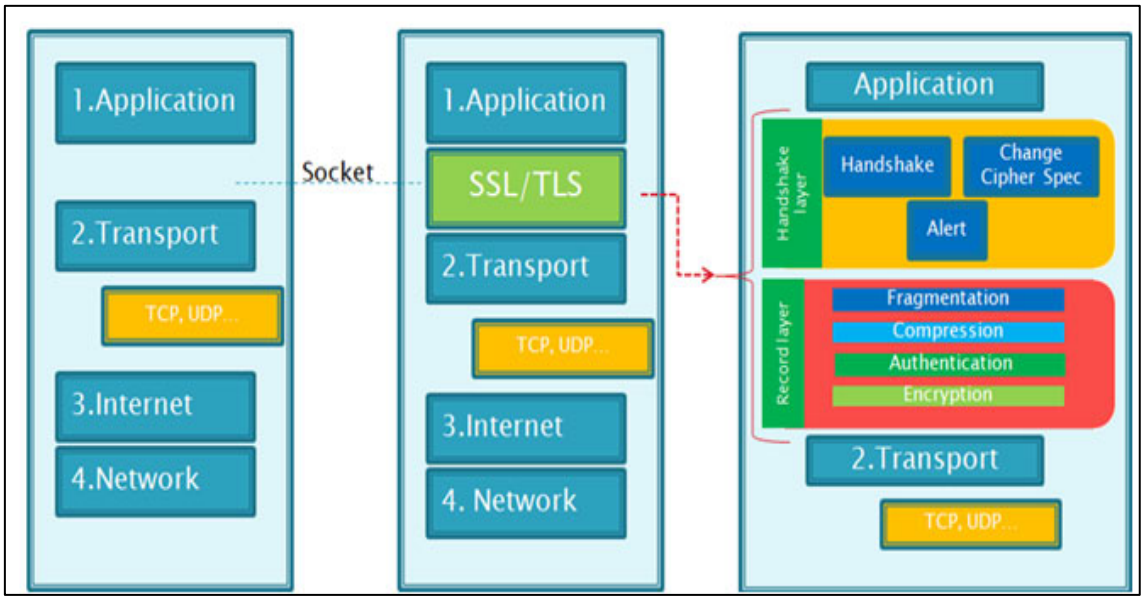


Figura 69. Estructura protocolo SSL/TLS

Los datos que se añaden como cabecera SSL que se envían no siguen la misma estructura en toda la comunicación, depende de la fase en la que esté la transmisión. La cabecera del protocolo es la siguiente.

Byte 0 Tipo de protocolo SSL enviado.

Byte 1-2 Especifica la menor y mayor versión SSL soportada por cliente y servidor

Byte 3-4 Longitud de la trama sin contar con la cabecera.

El byte 0 puede tener los siguientes valores:

Tipo de protocolo	Valor
Protocolo de cambio de cifrado	20
Protocolo de alerta	21
Protocolo de negociación	22
Protocolo de Registro	23

Tabla 2. Tipos de mensajes SSL/TLS

El resto de bytes son los datos de cada tipo de protocolo SSL enviado. El tamaño máximo de la trama sin sumar la cabecera es de 214 bytes.

Protocolo de Negociación

Este protocolo también conocido como *Handshake Protocol* se aplica en la primera fase y se encarga de establecer los parámetros para una conexión segura que se usaran para la transferencia de los datos. Los datos que se envían en el protocolo de negociación siguen el siguiente esquema.

Byte 5 Tipo de protocolo de negociación

Byte 6-8 Longitud de los datos del resto del protocolo

Byte 9-n Datos asociados al tipo de protocolo de negociación

Protocolo de Alerta

El protocolo de alerta envía un byte para especificar si la alerta ha sido de peligro o crítica y un segundo byte para informar del tipo de alerta. Dependiendo del tipo de alerta, la conexión puede verse afectada y desconectada para evitar que posibles intrusos puedan aprovecharse.

Protocolo de Cambio de Cifrado

Este protocolo envía un byte de valor 1 para informar a cliente y servidor que se tiene que cambiar el cifrado en la conexión establecida. La transmisión se pausa y empieza la renegociación evitando enviar los comandos innecesarios. Si la renegociación no se ha

llegado a completar, los parámetros de cifrado que se utilicen serán los mismos que se estaban utilizando.

Protocolo de Registro

El protocolo de registro es el encargado de enviar los datos según los parámetros de negociación. Los datos suministrados por las capas superiores son comprimidos, se le añade un MAC, después son encriptados y por último se le añade la cabecera para enviarlos a la capa de transporte.

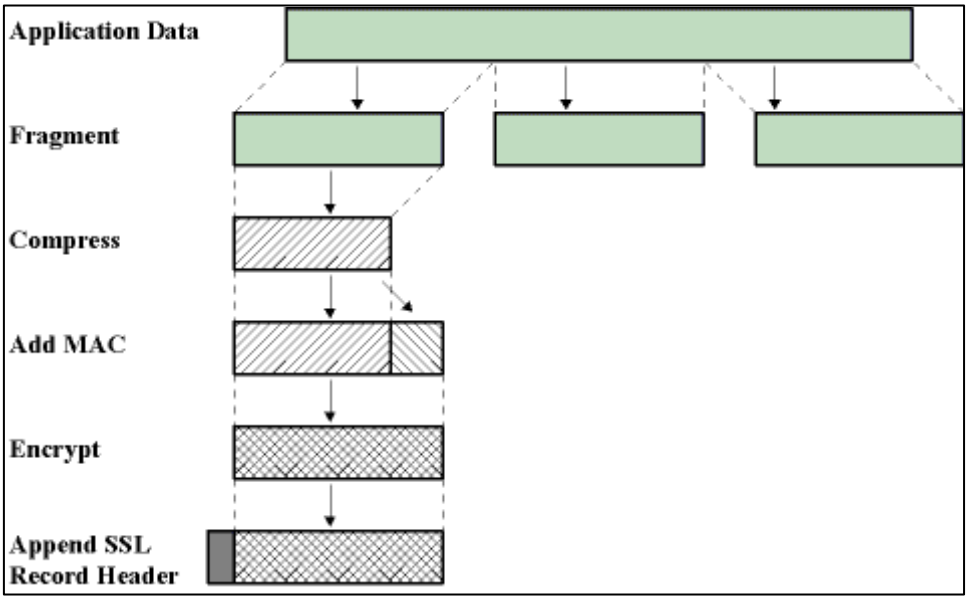


Figura 70. Encapsulado Protocolo SSL/TLS

El MAC, *Message Authentication Code*, es un conjunto bytes que se insertan al final de los datos para comprobar la integridad de los datos, es decir, para comprobar que los datos no han sido manipulados. El código de autenticación se genera combinando una función de encriptación *hash* con una llave de encriptación secreta. Hay dos tipos de algoritmos MAC, el MD5 que tiene una longitud de 16 bytes y el SHA-1 con una longitud de 20 bytes.

4.4.2. Secure SHel

El protocolo SSH se diseñó por Tatu Ylönen de la Universidad Tecnológica de Helsinki en 1995. En su primera versión el protocolo tenía vulnerabilidades graves por lo que la

segunda versión no tardo en salir a la luz. En la versión SSH2 se corrigieron los graves problemas y hubo mejoras en numerosos apartados. Actualmente la versión más usada es la SSH2 debido a los problemas que tiene la versión inicial.

El protocolo se creó con la intención de crear un sistema seguro para el intercambio de datos simple y barato de implementar. En un primer momento se enfocó en dar un acceso de autentificación remoto para reemplazar a telnet y otros sistemas de autenticación remotos, para ello, el SSH empezó a utilizar los comandos *rlogin, rsh* y *rcp* en la versión 1 de forma similar a los comandos que utilizaba telnet, en la versión 2 estos comandos fueron sustituidos por *slogin, ssh y scp.*

En la actualidad el grupo encargado del desarrollo del protocolo y lanzamiento de nuevas versiones como mejoras es el **IETF** (*Internet Engineering Task Force*), la última versión se recoge en los documentos RFCs del 4250 al 4256 dónde se especifica el estándar.

La función del protocolo es la creación de un túnel entre cliente-servidor previa autenticación de ambos, para que los datos viajen comprimidos y encriptados por el puerto 22. Para comprobar que los datos no han sido manipulados por el camino, se hace una comprobación del paquete cuando llega a su destino, si se detecta que el paquete ha sido manipulado, se transmitirá un mensaje de error y se procederá a la desconexión.

En el momento de establecer la comunicación, tanto cliente como servidor se autenticaran, para ello se realiza un intercambio de claves simétricas o claves públicas. A parte de establecer la autenticación, también se fijan los algoritmos para la encriptación de datos, la compresión y la verificación de los paquetes.

La forma de proceder del protocolo es muy similar al del protocolo SSL/TLS, cuando ya se ha realizar la negociación se procede al envió de los datos. Si se ha fijado un algoritmo de encriptación, los datos que provienen de las capas superiores se les aplicarán un algoritmo para que no puedan ser legibles, de esta forma se asegura la confidencialidad de los datos. Luego los datos se comprimen para reducir el tamaño,

haciendo eficiente la conexión. Por último, cada paquete es verificado por un algoritmo para comprobar que es el paquete que se envió.

A diferencia del protocolo SSL/TLS, el protocolo SSH puede establecer varias ventanas en una misma sesión, esto quiere decir, que se pueden enviar por el mismo canal varios datos a la vez de diferentes aplicaciones debido a que la autentificación entre cliente-servidor ya se ha realizado y permite la redirección de puertos TCP para la conexión.

4.4.2.1. Estructura del protocolo SSH

El protocolo SSH2 tiene una estructura similar a la del protocolo SSL/TLS, está dividido en dos niveles.

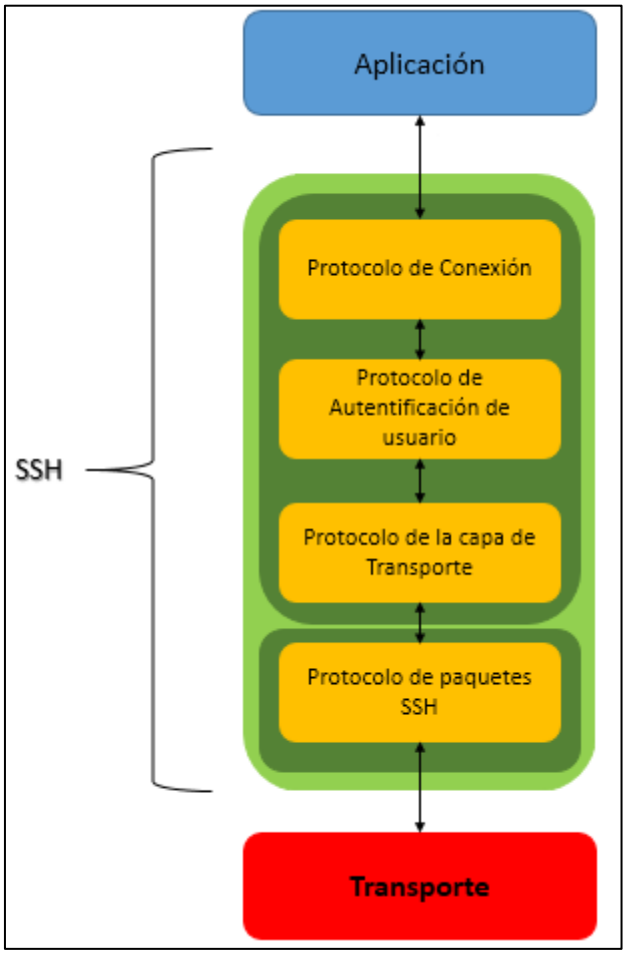


Figura 71. Estructura Protocolo SSH

En el primer nivel se sitúan tres protocolos, que son los encargados del tipo de mensaje a enviar. El primero de ellos es el Protocolo de Transporte, estando en el nivel más bajo dentro del protocolo SSH. Este protocolo proporciona un cifrado fuerte, una autenticación por parte del servidor, un cifrado de integridad y una opcional compresión de datos. El segundo protocolo es el Protocolo de Autentificación, para la autentificación por parte del usuario. Esta autentificación puede ser de dos tipos, con claves simétricas o con claves públicas. El último protocolo es el Protocolo de Conexión, se encarga de gestionar las sesiones interactivas y el redireccionamiento de puertos. En el segundo nivel se encuentra el Protocolo de Paquetes, que se encarga de la compresión y cifrado de los datos, añadido de MAC y envió de los datos por el túnel creado.

Protocolo de Paquetes SSH

Es el encargado de enviar los paquetes hacia la capa inferior. El proceso que realiza es la compresión para disminuir el volumen de los datos, añadir un código MAC al final del paquete y realizar un cifrado a todo el paquete menos a la parte del código MAC. En el destino, el proceso se tendrá que realizar de forma inversa para la obtención de los datos.

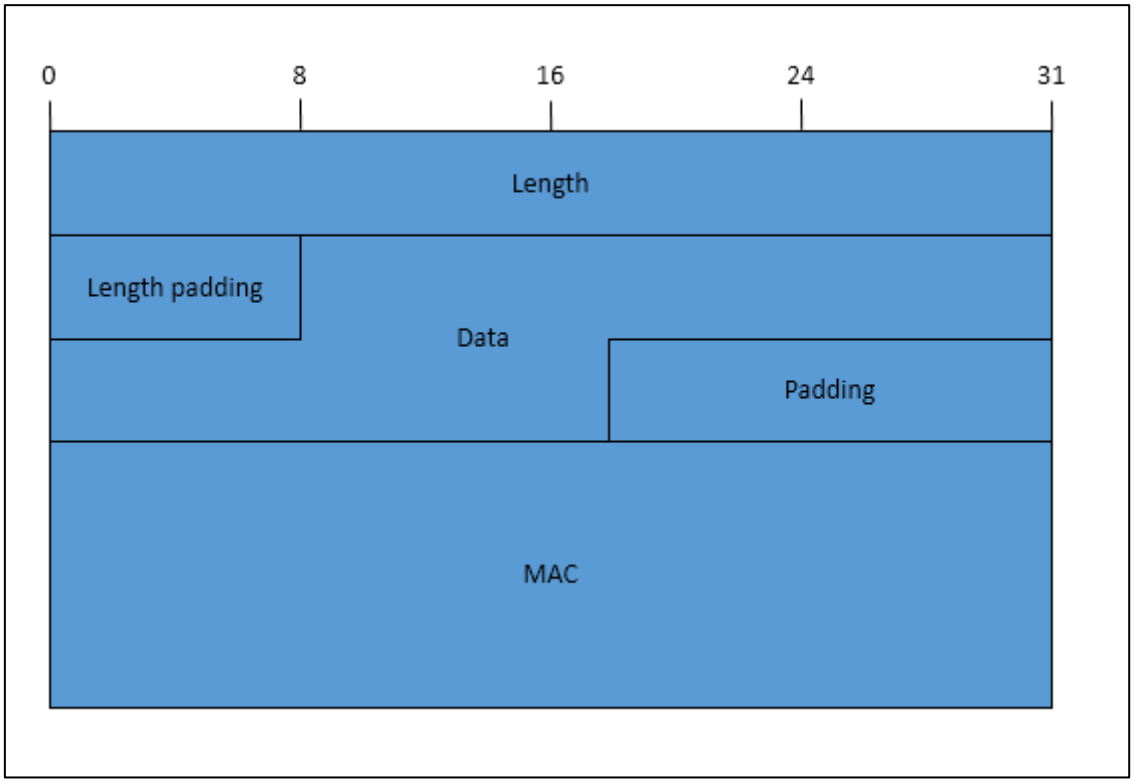


Figura 72. Estructura paquete SSH

La estructura del paquete está formada de la siguiente forma, siendo el tamaño máximo del paquete 35000 bytes.

- *Length (4 bytes).* Especifica la longitud de todo el paquete menos los bytes de la codificación MAC, es decir, la suma de $1+L_m+L_p$.
- *Length padding (8 bits).* Indica cuantos bytes hay de relleno (*padding*).
- *Data.* Los bytes pertenecen al mensaje, el primer byte indica el tipo de mensaje y los siguientes la información asociada al tipo de mensaje.
- *Padding.* Indica cuantos bytes de relleno se han puesto debido al cifrado de bloque, dependiendo del tipo de cifrado varia el campo hasta un máximo de 255 bytes y un mínimo de 4 bytes.
- *MAC (16-20 bytes).* Puede ser de longitud 0 bits si no se ha especificado ningún código MAC durante la negociación, 16 bytes si el código MAC es de tipo MD5 y 20 bytes si es de tipo SHA-1.

Protocolo de Transporte SSH

La función es similar a la del *Handshake Protocol* del protocolo SSL/TLS, se encarga de la negociación los algoritmos a utilizar cuando se establece la conexión. Se diseñó de forma que la negociación fuese simple y fluida, siendo posible en solo 2 intercambios de paquetes o como mucho 3. El cliente puede enviar paquetes con información de cifrado sin que el servidor haya hecho una petición.

En primer lugar se negocia cual es la versión a usar en la conexión, se elegirá la mayor que soporten tanto cliente como servidor. Después el servidor envía una clave simétrica que se usará en el proceso de intercambio de claves, normalmente de tipo *Diffie-Hellman*, para que los paquetes que se envíen con los algoritmos a utilizar en la conexión se trasmitan cifrados. El cliente deberá enviar su clave simétrica al servidor para cifrar los paquetes que se envíen en el intercambio de claves. Una vez que se han enviado estas claves, cliente y servidor se autentican mediante el protocolo de Autenticación, seguido del intercambio de los algoritmos de cifrado y compresión de datos y la clave MAC a incluir en cada paquete. Cuando se ha realizado el intercambio de claves satisfactoriamente, se empezarán a enviar los paquetes encriptados con los algoritmos negociados. Para indicar que el paquete SSH es de este tipo de protocolo, el primer byte del campo mensaje tiene que tener un valor entre 1 y 49.

Rango de valores	Tipo
1 al 19	Mensajes genéricos del Protocolo de Transporte.
20 al 29	Negociación de algoritmos.
30 al 49	Intercambio de claves con métodos específicos (los números puede ser reusado por diferentes métodos de autentificación).

Tabla 3. Tipo de mensajes Protocolo de Transporte SSH

Protocolo de Autentificación

Una vez que se ha fijado la clave simétrica para el intercambio de claves, es turno para la autenticación del usuario. Cuando se ha recibido el identificador de la sesión del Protocolo de Transporte, el servidor envía un listado de los métodos de autenticación

al cliente que soporta, el cliente envía un mensaje de autentificación basado en una clave privada para su autentificación. El cliente tendrá un periodo de 10 minutos o 20 intentos de autentificarse de forma correcta, sino el servidor cerrara la conexión. Aunque no es obligatorio, el cliente puede establecer una conexión sin autentificarse siempre que el servidor permita este tipo de autenticaciones.

Los números de tipo de mensaje para este protocolo tiene el rango del 50 al 59 y del 60 al 79.

Rango de valores	Tipo
50 al 59	Mensajes genéricos del Protocolo de Autentificación.
60 al 79	Mensaje de autentificación para especificar el método (los números puede ser reusado por diferentes métodos de autentificación).

Tabla 4. Tipo de mensajes Protocolo de Autenticación SSH

Protocolo de Conexión

Este protocolo gestiona todas las conexiones que puede haber entre un cliente y un servidor. Crea un túnel en el cual multiplexa todas las conexiones lógicas en el origen, en el destino se demultiplexa todas las conexiones.

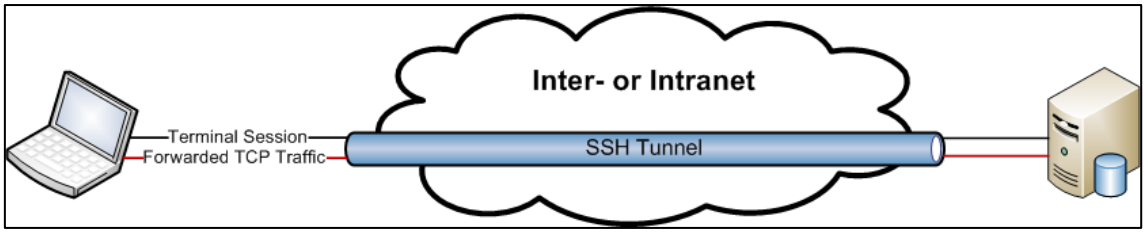


Figura 73. Protocolo de Conexión SSH

Hay 4 tipos de canales lógicos:

- *direct-tcpip.* El cliente especifica el puerto que usará en la conexión.
- *forwarded-tcpip*. El cliente indica al servidor el puerto que tiene que utilizar en la conexión.

- *Redireccionamiento X11.* Permite ejecutar aplicaciones gráficas de un servidor remoto en nuestro escritorio.
- *Session*. Permite la ejecución de un programa en el equipo remoto.

Los valores del primer byte del campo mensaje para esto protocolo tiene un rango del 80 al 127.

Rango de valores	Tipo
80 al 89	Mensajes genéricos del Protocolo de Conexión.
90 al 127	Mensajes relacionados con el canal.

Tabla 5. Tipos de mensajes Protocolo de Conexión SSH

4.4.3. Algoritmos de cifrado

Uno de los problemas en las comunicaciones es la seguridad, cualquier equipo que este dentro de la red utilizada puede leer los datos que se transfieren e incluso enviar el mismo datos engañando a los demás equipos. Por esta causa se empezó a cifrar los datos para que estos, aunque puedan ser interceptados por un equipo no incluido en la conexión, no pueda acceder a ellos, solo el origen y el destino conocen el algoritmo de cifrado y descifrado para proporcionar una conexión segura.

Todos los algoritmos están creados a partir de funciones complejas que los ordenadores convencionales tardarían miles de años que averiguar la clave. Aunque haya algoritmos que a día de hoy sean indescifrables, es aconsejable regenerar la clave cada año por precaución, además los algoritmos que hay en la actualidad se mejoran día a día ofreciendo más robustez.

Las claves de cifrado pueden ser de dos tipos, simétricas, tanto el origen como el destino tiene la misma clave para cifrar y descifrar los datos o asimétrica, el origen tiene una clave y el destino tiene otra, cuando el origen cifra un paquete lo hace de manera que solo la clave de destino pueda descifrarlo, por lo tantos estas claves de generan a la vez y dependen la una de la otra. En los protocolos anteriores, SSL/TSL y SSH, se utilizan diferentes cifrados dependiendo de la fase de la conexión.

Algoritmo de intercambio de claves

Este algoritmo se establece al principio de la comunicación, cuando el canal no es seguro ni hay cifrado en los paquetes de la capa de aplicación, por lo que está expuesto a ataques de tipo *man-in-the-middle*, en el que un equipo ajeno a la comunicación que puede interceptar los datos. Hay dos tipos de algoritmos, el ***Diffie-Hellman***, un cifrado de tipo simétrico que es invulnerable a ataques de tipo *man-in-the-middle* utilizado por SSL/TLS y SSH y el algoritmo **RSA**, algoritmo de clave pública, el cliente tiene una clave privada y el servidor una clave pública que no ha sido enviada por el medio, solo con estas claves puede descifrar los paquetes que se envían.

Algoritmos de autenticación

Los algoritmos de autenticación permiten validar al usuario y al servidor para establecer la comunicación entre dos equipos. El servidor tiene la obligación de autenticarse, en el caso del protocolo SSL/TLS en vez de autenticarse con un algoritmo, puede autenticarse con un certificado digital (CA), basado en un certificado **X.509**. Tanto el protocolo SSL/TLS como el SSH pueden utilizar algoritmos asimétricos **RSA** y **DSA**.

Algoritmos de encriptación

Para evitar que los paquetes sean capturados por un equipo no aceptado en la comunicación y pueda leer los datos que se envíen, se usa un cifrado de encriptación. Después de añadir los bytes de autentificación del Mensaje (MAC) al final del paquete, los datos se codifican de manera que no puedan ser legibles, solo el destino que conoce el algoritmo de cifrado de paquetes utilizado en la transferencia puede descifrar el paquete y leer los datos contenidos en él. En el caso del protocolo SSL/TLS se cifran todos los datos, incluyendo el campo MAC, mientras que en el protocolo SSH se cifran todos los datos menos este campo. Los tipos de cifrados que hay en ambos protocolos, SSL/TLS y SSH, son **RC4**, **3DES** y **AES** de tipo simétrico. A parte el protocolo SSH permite los cifrados **DES**, **Blowfish**, **Twofish**, **Serpent**, **IDEA** y **CAST-128** de tipo simétrico.

Algoritmos de compresión

Este algoritmo se utiliza para mejorar la eficiencia del envió de datos. Debido a la compresión, los datos tienen un tamaño menor y los paquetes son más pequeños, con esto se consigue no sobrecargar el medio. La compresión de los datos se realiza en el primer momento, cuando le llegan los datos de la capa superior. En SSL/TLS utiliza un algoritmo propio del protocolo mientras que el protocolo SSH utiliza el algoritmo **GZIP**.

Algoritmos de autentificación del mensaje

El algoritmo de autentificación, también llamado **MAC** (*Message Cuthentication Code*), se encarga de verificar los paquetes en el destino, para comprobar si el paquete ha sido manipulado. Este algoritmo utiliza una función *hash* con una clave secreta **K** que solo el cliente y servidor conocen para crear una firma digital que se añade al final del paquete después de la compresión de los datos de la capa superior. Hay dos tipos de algoritmos tanto en SSL/TLS como en SSH, el **SHA-1** y el **MD5**, ambos algoritmos asimétricos.

5. SIMULACIÓN DE UNA RED VSAT CON DVB-S/RCS

A lo largo del proyecto se ha explicado de forma teórica todos los conceptos necesarios para el correcto funcionamiento de una red VSAT bajo la norma DVB-S/RCS. Ahora se va a proceder a desarrollar una aplicación que muestre de una manera visual el comportamiento real de estas redes, llamada “Simulador de DVB-S”.

Se ha usado como modelo de referencia la Red SAICA (Sistema Automático de Información de Calidad de las Aguas), figura 74, empleada por la Confederación Hidrográfica del Tajo para el control de la calidad del agua en el río Tajo en tiempo real. La red está compuesta por una serie de estaciones situadas a lo largo del río y una estación central ubicada en el centro de control de la red, en este caso en Madrid. Las estaciones remotas se encargan de tomar las medidas de ciertos parámetros relacionados con la calidad del agua (temperatura, oxigeno, conductividad, acidez…) en intervalos de quince minutos, creando ficheros de texto con un formato establecido y conocido por todas las estaciones de la red, formato SAICA. Posteriormente estos ficheros son enviados a la estación central para su procesamiento. La comunicación entre las estaciones remotas y la central se produce a través de enlaces satelitales, en algunos casos se emplean radioenlaces, debido a que en la situación geográfica donde se encuentran las estaciones no existe una infraestructura terrestre, formando una red VSAT.

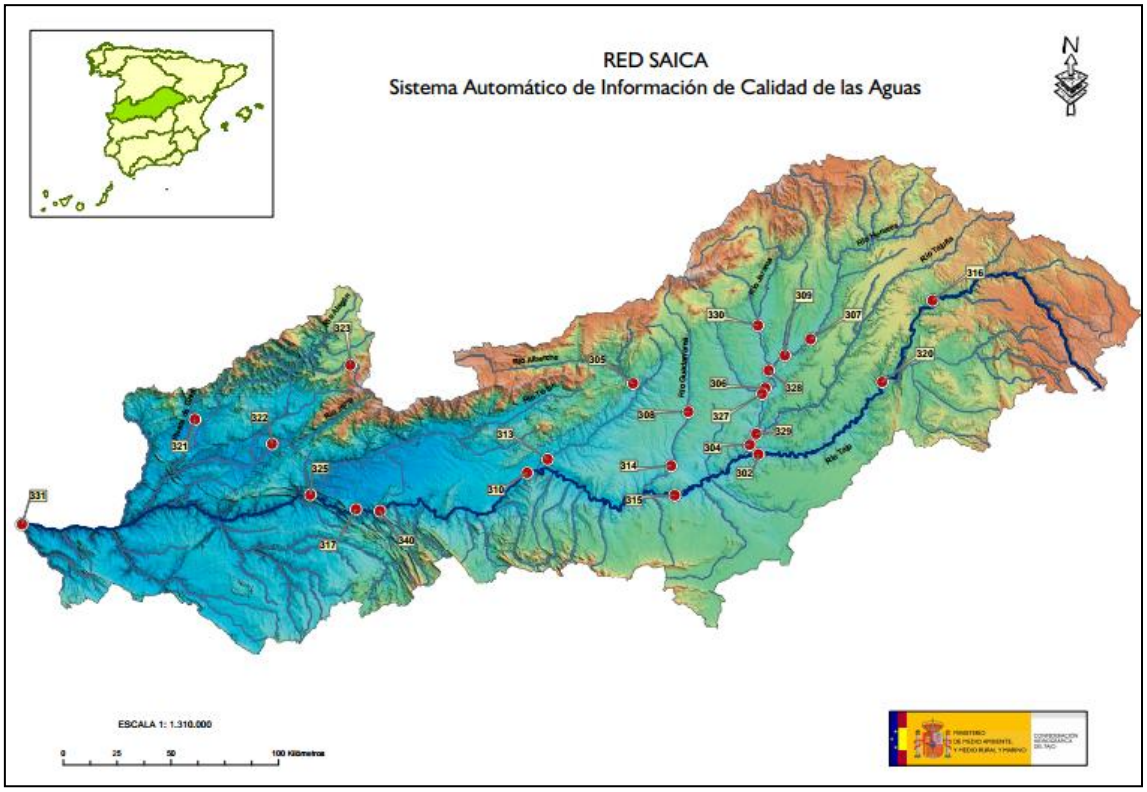


Figura 74. Mapa red SAICA

5.1. "Simulador de DVB-S"

El programa ha sido diseñado con el objetivo de ser una herramienta didáctica que enseñe el funcionamiento real de un sistema VSAT trabajando con el estándar abierto DVB-S. Muestra los diferentes protocolos por los que pasa el mensaje (archivo), desde la creación del mismo hasta su transmisión, permitiendo ver el valor de los campos de cada uno de ellos así como una breve descripción del cometido de cada uno de ellos.

Además de su función didáctica, la aplicación también tiene una parte práctica. Gracias a la posibilidad de cargar un archivo que contenga los datos en formato SAICA, y a la forma en la que se representan los mismos, es fácil comprobar si ha habido un error sin tener que examinar la cadena de dígitos manualmente separando cada uno de los campos. Esto es de gran utilidad, por ejemplo, para un técnico que vaya a hacer una inspección de una estación remota. Simplemente accediendo al directorio donde estén almacenados los archivos, podría cargarlos en la aplicación y comprobar si ha habido algún error a la hora de tomar las medidas y si la paramétrica funciona correctamente.

5.1.2. Manual de usuario del "Simulador de DVB-S"

El siguiente capítulo tiene como objetivo explicar el correcto funcionamiento del programa "Simulador de DVB-S", formado por tres estaciones conectadas vía satélite.

1. Instalación del simulador

El programa ha sido diseñado para funcionar en cualquier versión de Windows sin necesidad de instalar ningún software, lo que otorga la ventaja de poder ser transportado en una memoria externa y ejecutado en cualquier lugar. Para arrancar el programa basta con hacer doble "*click*" sobre el archivo ejecutable.

2. Ventana principal

Al iniciar el programa aparece la ventana principal, que tiene un aspecto como el que se muestra en la figura 75.

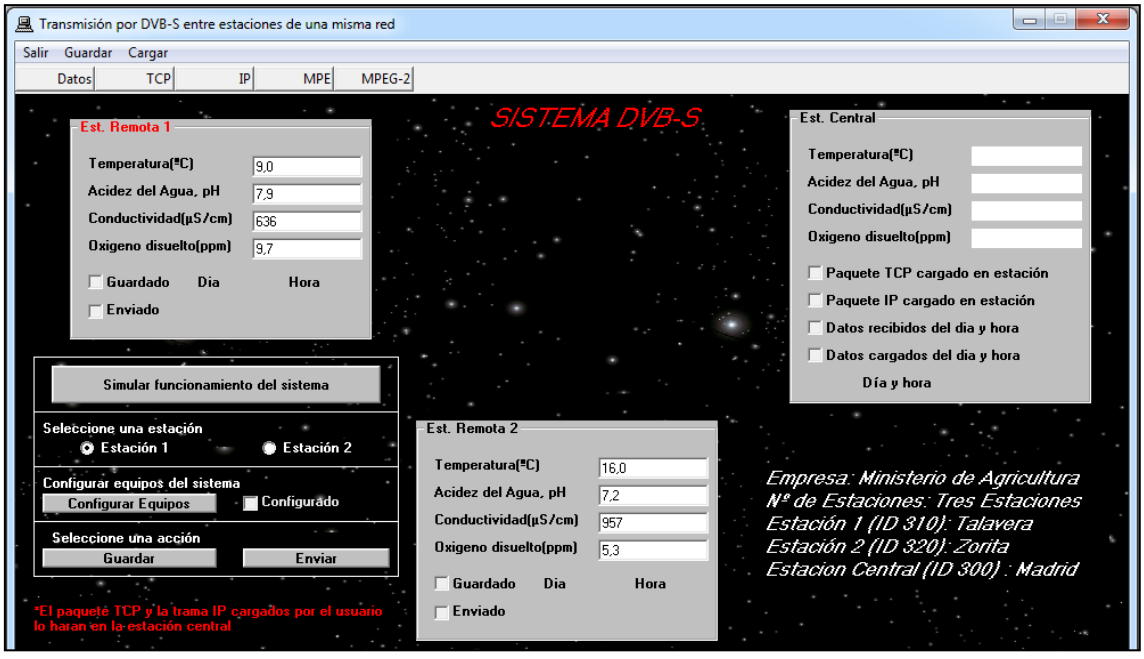


Figura 75. Ventana principal Simulador de DVB-S

En esta ventana aparecen las tres estaciones que forman parte de nuestra red, dos estaciones remotas y una estación central, y un panel de control desde donde manejar el programa. Además de un menú y una barra de herramientas.

2.1 Estaciones Remotas

Las estaciones remotas son estaciones de medida situadas a lo largo del río Tajo, concretamente en este caso en Talavera de la Reina (Toledo) y en Zorita (Guadalajara). Las figuras 76 y 77 muestran dichas estaciones.

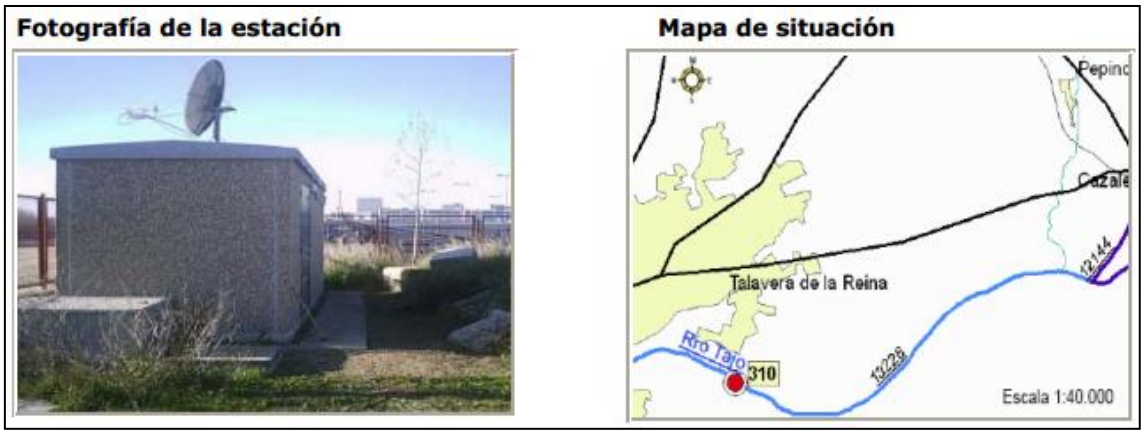


Figura 76. Estación remota de Talavera de la Reina (Toledo)

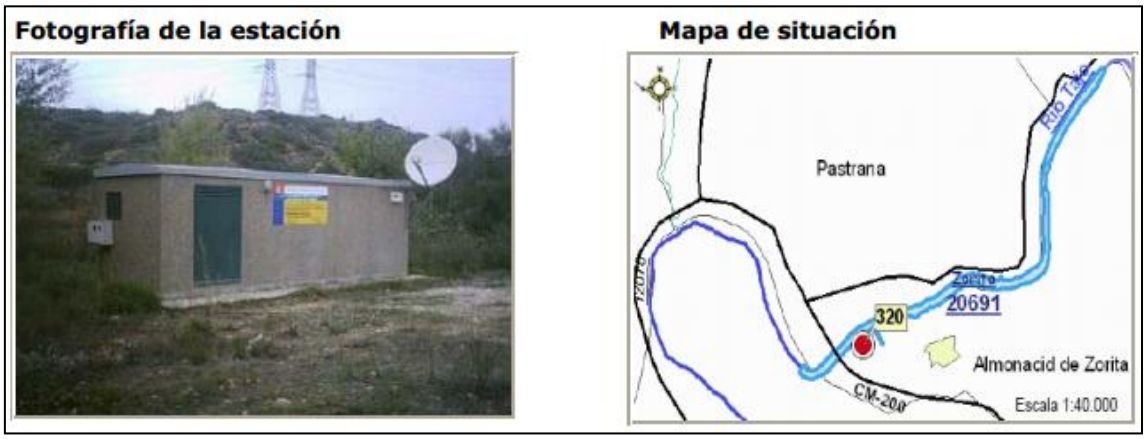


Figura 77. Estación remota en Zorita (Guadalajara)

En el programa, las estaciones han sido representadas mediante una caja que muestra los valores de los parámetros medidos así como si los datos han sido guardados y enviados a la estación central, tal y como muestra la figura 78.

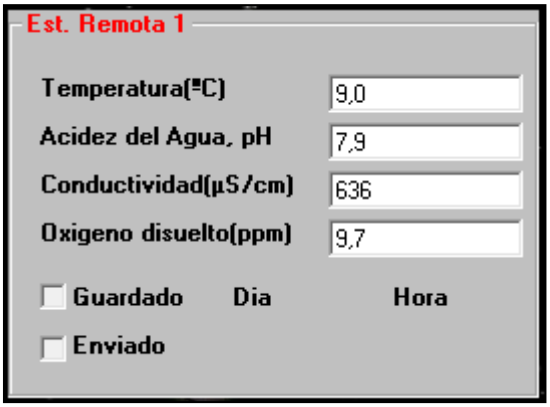


Figura 78. Parámetros de la calidad del agua

Las estaciones escogidas para esta recreación animada están dotadas de una multiparamétrica capaz de tomar cuatro medidas (Temperatura, Acidez, Conductividad y Oxigeno) con una precisión de un decimal. La figura 79 representa el típico analizador multiparamétrico con el que cuentan las estaciones.

Figura 79. Analizador multiparamétrico

2.2 Estación Central

La estación central es la encargada de recoger y almacenar los datos tomados por las diferentes estaciones remotas para su posterior evaluación. En este caso se encuentra en Madrid, que es donde está el centro de control. La figura 80 muestra una fotografía de la estación central de Madrid.

Figura 80. Estación central de Madrid

En la aplicación, la estación central se representa mediante una caja que permite ver las medidas tomadas por las estaciones remotas indicando la hora y el día exacto en que fueron hechas, figura 81. También permite saber si esos datos han sido cargados por el usuario mediante un archivo ya existente en la base de datos o son los recibidos en ese momento. Los datos pueden ser cargados de tres maneras diferentes, a través del archivo de datos SAICA, del paquete TCP o el paquete IP.

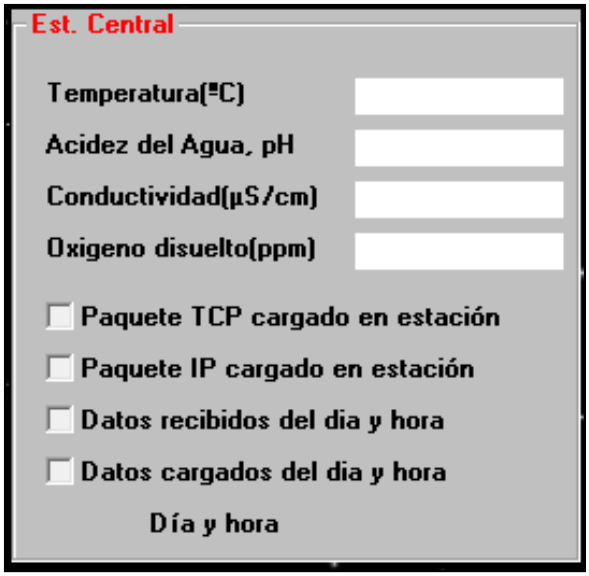


Figura 81. Datos estación central

2.3 Panel de control

El panel de control del programa, figura 82, controla el comportamiento del sistema. Consta de varios elementos que se pasan a detallar.

- **Simulación del sistema.** Recrea el comportamiento del sistema de forma animada.
- **Selección de estación:** Indica sobre que estación se está actuando.
- **Configuración de los equipos.** Permite entrar dentro de cada terminal para configurar sus parámetros básicos como son el número de puerto, la dirección IP y la dirección MAC.
- **Selección de acción.** Permite elegir entre guardar los datos tomados por las estaciones o enviar los datos ya guardados.

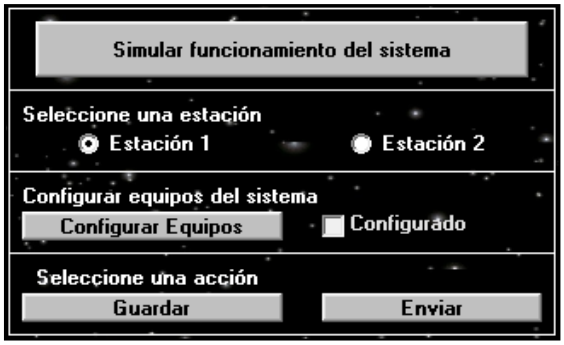


Figura 82. Panel de control

3. Menú

El menú, situado en la parte superior izquierda de la ventana principal, ofrece tres opciones:

- **Salir**. Cierra la aplicación.
- **Guardar**, figura 83. Permite guardar los datos de las estaciones en un archivo de tipo texto con el nombre que se quiera y en un directorio diferente al que viene por defecto. Se puede elegir entre guardar los datos en formato SAICA, el Paquete TCP o el Paquete IP.

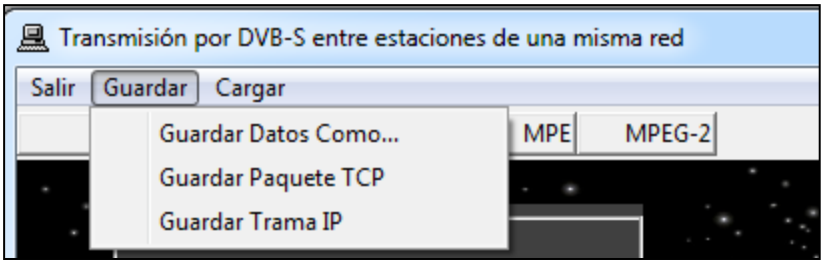


Figura 83. Opción guardar del menú superior

- **Cargar**, figura 84. Permite cargar un fichero en una estación. Los datos se pueden cargar de tres maneras diferentes: cargando un archivo con formato SAICA, un paquete TCP o un paquete IP.

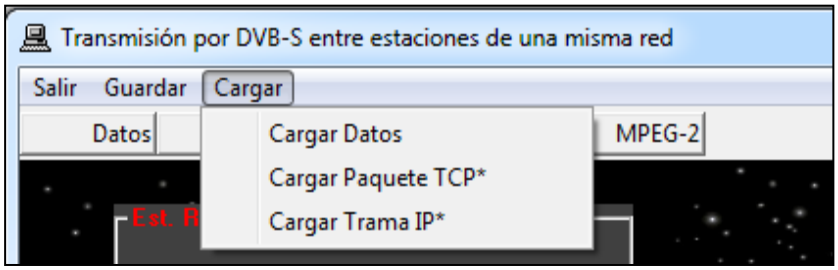


Figura 84. Opción cargar del menú superior

4. Barra de herramientas

La barra de herramientas, figura 85, se encuentra en la venta principal y ofrece las distintas posibilidades de visualizar el contenido cargado en el programa.

Figura 85. Barra de herramientas

4.1 Datos

Esta opción permite ver los datos almacenados en una estación, figura 86. Lo muestra de tal manera que permite hacer una rápida comprobación de si hay algún error de medida o del tipo que sea en el archivo de datos.

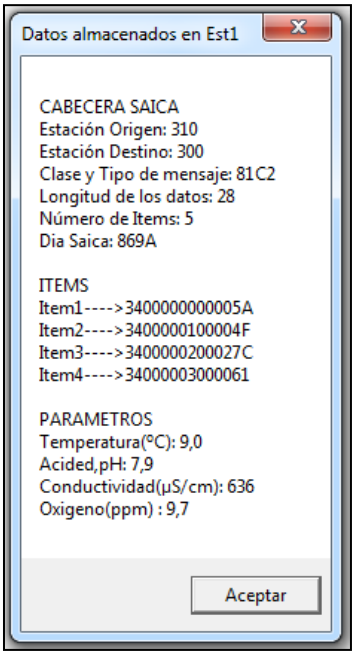


Figura 86. Datos de una trama SAICA

Como se puede ver, la representación de los datos está estructurada en tres partes, en donde la primera representa el valor de la cabecera SAICA, la segunda el valor de los *Items* de cada una de las medidas tomadas y por último el valor de cada sensor de la paramétrica, este último extraído de la información de los *Items* anteriores.

4.2 TCP

La opción TCP muestra en detalle el paquete TCP de la estación elegida, figura 87.

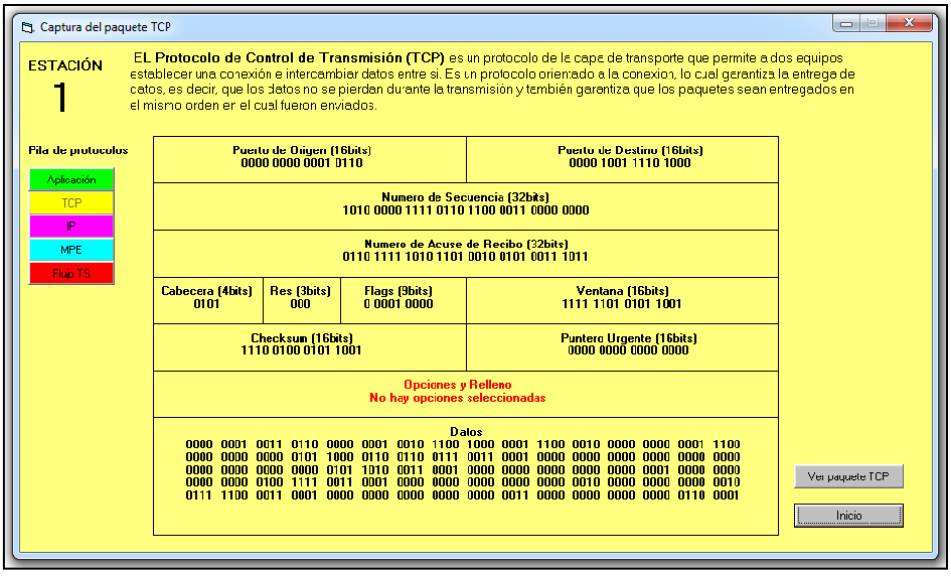


Figura 87. Paquete TCP

Como se aprecia en la figura anterior, hay una breve descripción del protocolo antes de pasar a hacer una representación de los campos que forman dicho paquete. Dentro de esta ventana hay varias opciones:

- **Pila de protocolos**, figura 88. Permite moverse directamente por los diferentes protocolos sin ir a la ventana principal. Cada protocolo está representado con un color diferente que ayuda a su identificación dentro de la pila.

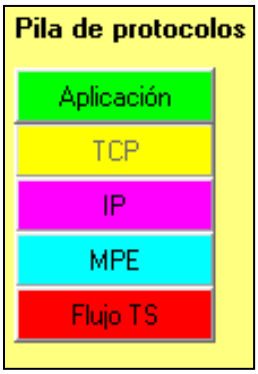


Figura 88. Pila de datos

- **Ver paquete TCP**, figura 89. Al hacer "*click*" en este botón aparece una ventana emergente que muestra el valor del paquete TCP tal cual, con sus valores en hexadecimal.

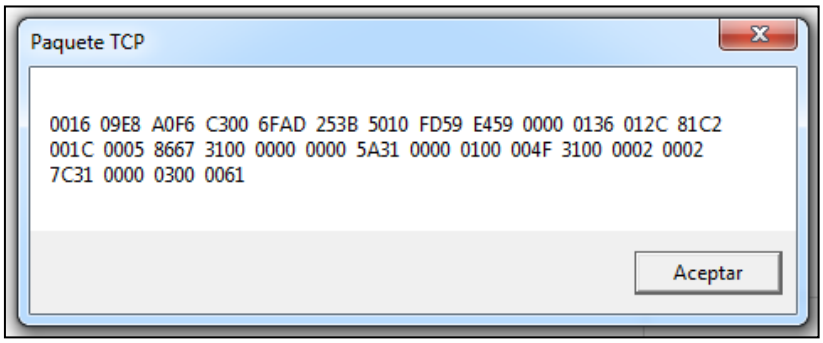


Figura 89. Paquete TCP

- **Inicio,** figura 90. Cierra la ventana actual y abre la ventana principal del programa.

Figura 90. Botón de inicio

El resto de opciones del panel de navegación (IP, MPE y MPEG-2) sigue una estructura idéntica a esta.

4.3 IP

Esta opción muestra el datagrama IP de la estación elegida, mostrando en detalle cada uno de sus campos, figura 91.

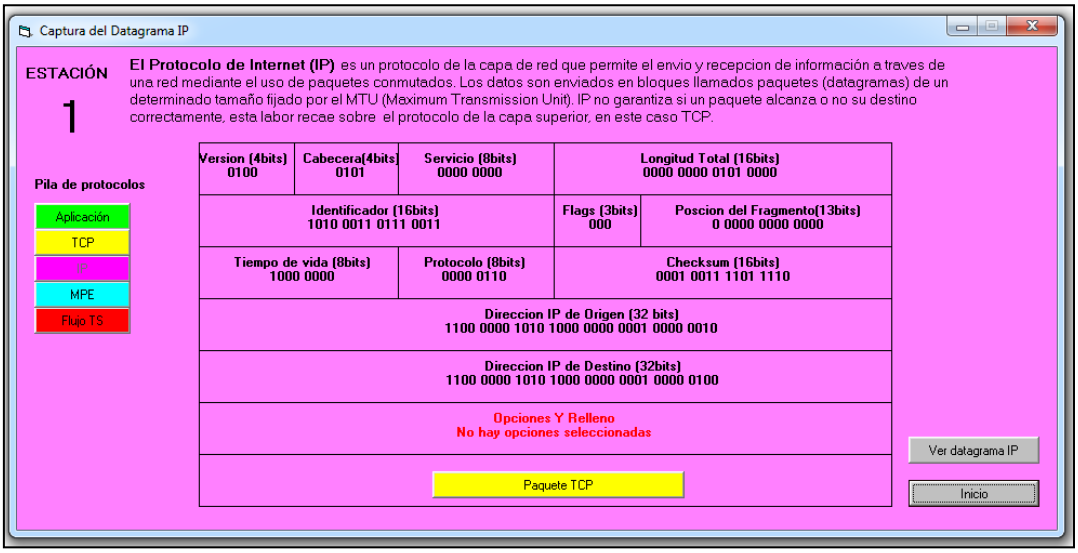


Figura 91. Paquete IP

4.4 MPE

Muestra el paquete MPE creado para poder enviar el paquete IP dentro del flujo de transporte MPEG-2.

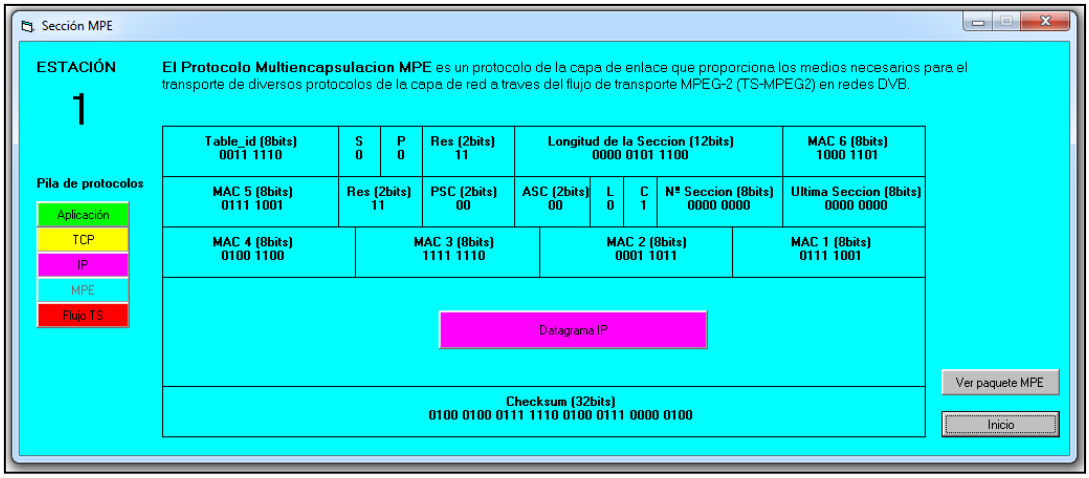


Figura 92. Sección MPE

4.5 MPEG-2

Muestra el paquete de transporte MPEG-2 del flujo de transporte. Es el que se manda a la estación central y contiene toda la información necesaria.

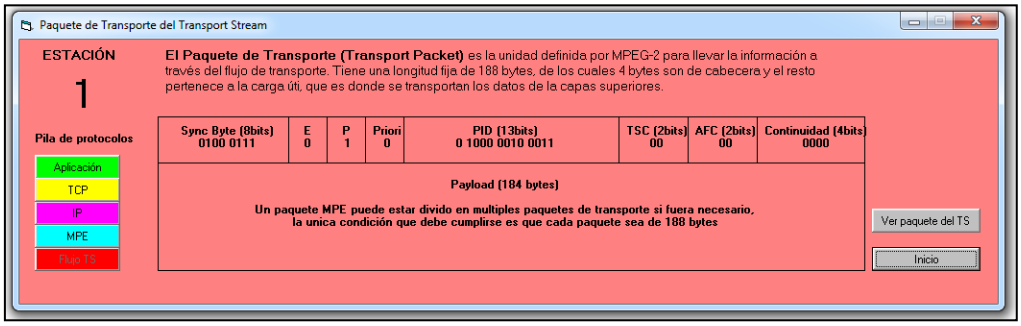


Figura 93. Paquete de Transporte de *Transport Stream*

5. Manejo del programa

El programa ha sido diseñado para trabajar en dos modos: **modo animado** o **modo manual**.

5.1 Modo animado

Se habilita cuando se presiona el botón **"Simular funcionamiento de sistema"**, recreando el comportamiento de la red de una forma animada. Éste muestra solo un ciclo de intercambio de información, pero que en la realidad se repetirá periódicamente.

Se puede apreciar como las estaciones remotas toman las medidas y las almacenan en un archivo de un directorio especial. Más tarde, la estación central manda una petición de datos a cada estación remota con el fin de descargarse todos los archivos nuevos desde la última conexión. Estas le responden mandándole los archivos a través del flujo de transporte definido por la norma DVB-S, el flujo MPEG-2. Una vez que la estación central ha recibido los datos, y no ha habido ningún error en la transmisión, ésta manda un mensaje de confirmación a cada estación indicando que todo ha ido bien.

Durante el funcionamiento de este modo, el **programa queda inhabilitado** para el usuario, teniendo que esperar que la animación finalice.

5.2 Modo manual

Este es el modo de funcionamiento principal del programa, siendo el elegido por defecto cuando arranca la aplicación. Aquí se puede introducir el valor de cada parámetro manualmente y ajustar la configuración de cada equipo según el usuario desee. El modo de proceder es el siguiente.

1. **Seleccionar la estación** donde se desea trabajar a través del panel de control o haciendo "*click*" sobre ella.
2. **Introducir** el valor de los **parámetros** Temperatura, Acidez, Conductividad y Oxigeno.
3. Guardar los valores introducidos mediante el botón **"Guardar"** disponible en cada una de las estaciones remotas. Esto genera un fichero con los datos almacenados en formato SAICA cuyo nombre es la fecha y hora en la que se ha creado. Cada estación guarda sus archivos en un directorio diferente.
4. Configurar los equipos mediante el botón **"Configurar equipos"**. Éste abre una nueva ventana, figura 94, donde poder ajustar los parámetros básicos de cada equipo. Si se ha configurado correctamente aparece un "*tick*" en la etiqueta **"Configurado"** del panel principal. Los parámetros que el usuario puede modificar son el puerto de entrada/salida, la dirección IP y la dirección MAC de cada equipo. No obstante, el puerto de entrada/salida de las estaciones remotas no se puede modificar ya que el protocolo SFTP (*Secure File Transfer Protocol*), que es el que lleva a cabo la transmisión para trabajar de forma segura, trabaja en el puerto 22 de TCP.
5. Por último, para enviar la información a la estación central solo falta dar al botón de **"Enviar"** de la estación correspondiente. Éste manda el archivo guardado anteriormente a la estación central, quien se encarga de mostrar los datos recibidos y de almacenarlos para su posterior estudio.

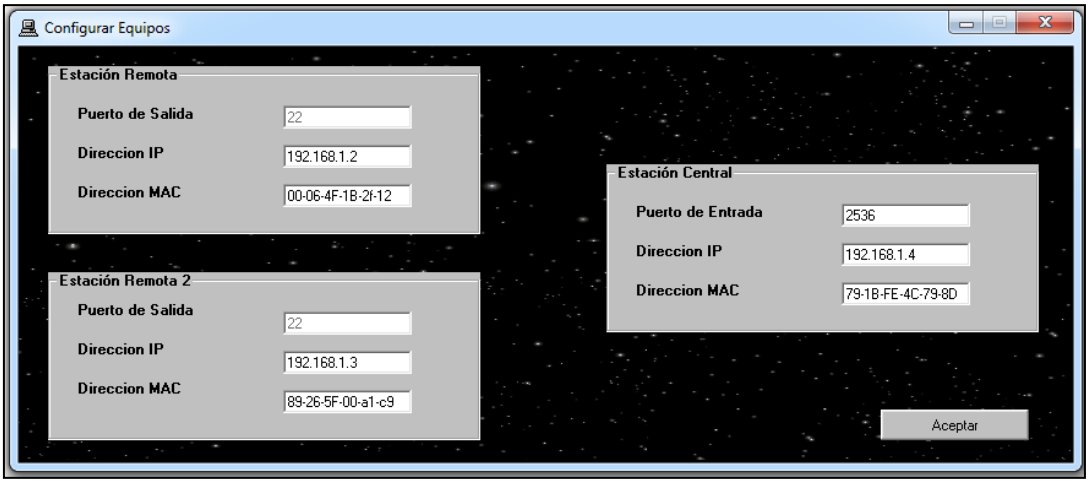


Figura 94. Ventana de configuración de equipos

6. TRANSFERENCIA DE FICHEROS EN UNA RED DVB-S/RCS

El FTP es un Protocolo de Transferencia de Ficheros entre un servidor y un cliente. Este protocolo actúa en la capa de aplicación del modelo TCP/IP utilizando por norma general los puertos 20 y 21, aunque se puede utilizar en cualquier puerto siempre y cuando en la configuración del servidor se defina.

Se diseñó con la idea de que entre cliente y servidor hubiese la máxima velocidad de conexión y así la descarga de ficheros fuese más rápida, pero no para que tuviese la máxima seguridad, lo que hace que sea un protocolo pobre en este aspecto. Hay variantes como el **SFTP** (*SSH File Transfer Protocol*) o el **FTPS** (*File Transfer Protocol over SSL*) que dotan a la conexión de más seguridad con el cifrado de datos desde ambos extremos.

Además también sufre carencias de seguridad en la parte del servidor, este puede ser atacado en el puerto 21 al ser muy vulnerable con ataques tipo *"spoofing",* este tipo de ataques lo que hacen es la suplantación de identidad del cliente para poder entrar en el servidor FTP de manera intrusiva.

6.1. *File Transport Protocol*

El cliente FTP inicia la conexión por el puerto del servidor FTP, por defecto será el 21 si el servidor no ha especificado otro puerto a la hora de configurarlo. En esta primera conexión cliente y servidor establecen los criterios para la conexión, como son los parámetros para la conexión y el tipo de datos que ambos van a manejar. Los parámetros de conexión que se establecen en la primera conexión serán:

- Puerto de datos
- Modo de transferencia
- Tipo de representación y estructura

También se especifican el tipo de acciones que el cliente FTP puede realizar en el servidor FTP.

- Almacenar
- Recuperar
- Añadir
- Borrar
- Obtener

Cuando ya se hayan establecido ambos parámetros, puede empezar la transferencia de datos, ya sea de cliente a servidor o viceversa.

6.2. Modo de conexión

La conexión entre cliente y servidor puede ser de dos tipos: modo activo o modo pasivo.

6.2.1. Modo activo

El cliente se conecta a través de un puerto de control aleatorio mayor al 1024 hacia el servidor por el puerto 21 o el que se haya configurado en el servidor previamente a la conexión, y le comunica al servidor cual es el puerto que el cliente va a utilizar para la transferencia de datos.

El servidor empieza la transferencia de datos por su puerto 20 hacia el cliente por el puerto que se le ha indicado anteriormente, de ahí que sea modo activo, porque es el servidor quién inicia la transmisión de datos. Se puede ver este tipo de conexión en la siguiente imagen.

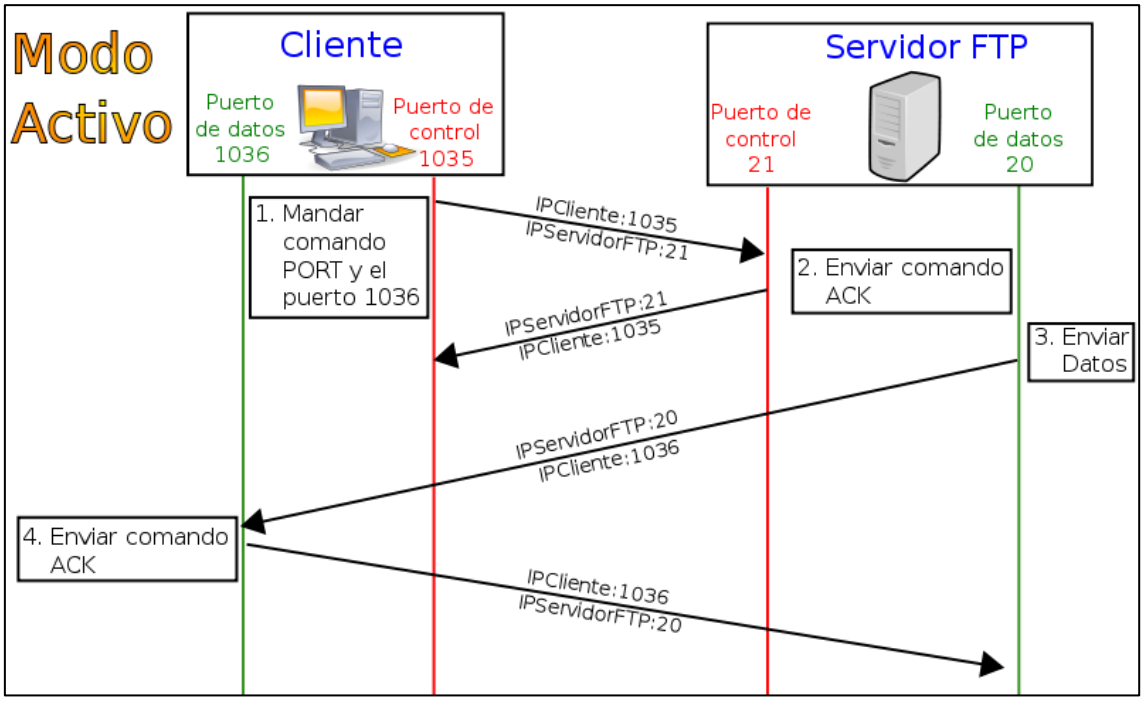


Figura 95. Modo activo FTP

El cliente inicia la conexión con el servidor por su puerto 1035 y le comunica al servidor que el puerto que quiere utilizar para la transferencia de datos es el 1036. El servidor acepta la conexión y seguidamente hace la conexión da datos por el puerto que el cliente le ha comunicado anteriormente.

6.2.2. Modo pasivo

En este modo solo hay un tipo de conexión, el cliente mediante el puerto de control, un puerto aleatorio mayor que 1024, se comunica con el servidor por su puerto 21, o el que se haya configurado previamente, y le pide que abra un puerto aleatorio mayor que 1024 para la conexión de datos. El servidor le contesta con la aceptación de la conexión y el puerto que ha abierto para la transferencia de datos, ahora es el cliente quién inicia la transferencia de datos por el puerto de control+1 hacia el puerto del servidor que este le ha indicado.

En la siguiente imagen se ve claramente el proceso de conexión empleado en este modo.

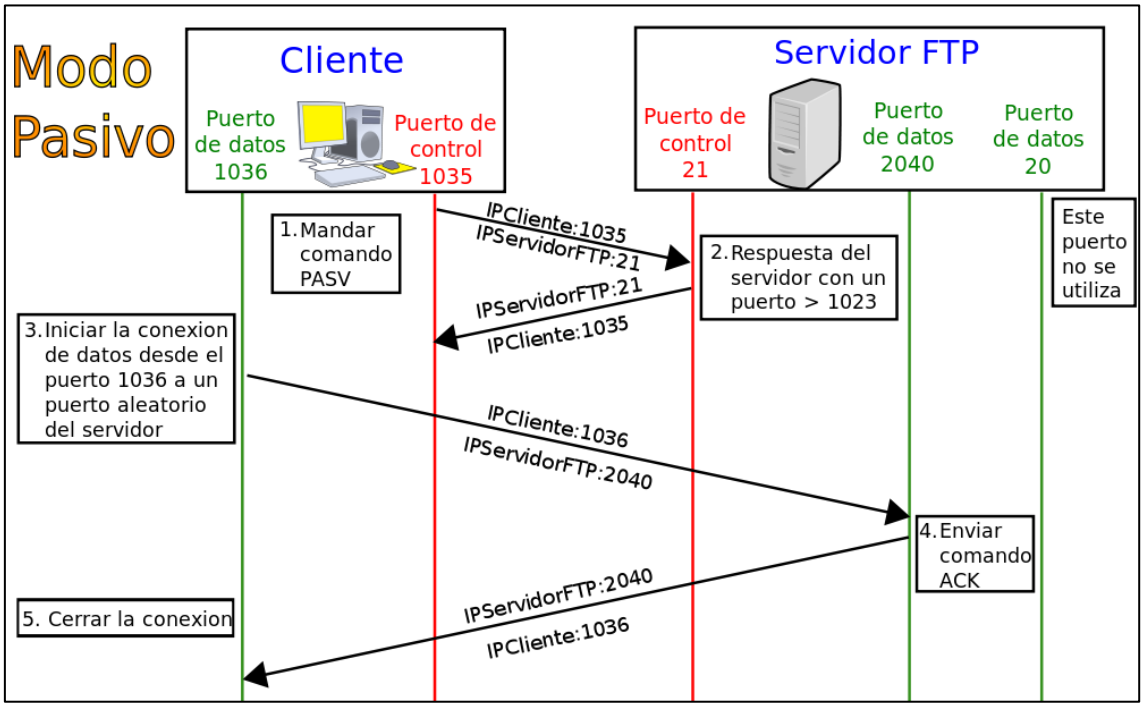


Figura 96. Modo pasivo FTP

El cliente envía un comando PASV al servidor indicándole el modo de conexión, el servidor le responde y el cliente inicia la transferencia de datos desde el puerto siguiente al puerto de control hacia el servidor por el puerto que este le ha indicado anteriormente.

En estos dos tipos de modos de conexión hay una diferencia, en el modo activo el servidor es quién inicia la conexión de datos y en el pasivo es el cliente, por lo que en el modo pasivo, si se tiene un cortafuegos puede que no se pueda tener una conexión entrante desde fuera.

6.3. Modos de acceso al servidor

Para conectarse al servidor hay diferentes tipos de conexión, cada uno cuenta con diferentes permisos para el manejo de datos y las carpetas que puedan visualizar. El acceso al servidor FTP se especifica con un usuario y una contraseña o sin estos parámetros.

6.3.1. Acceso anónimo

Este tipo de acceso no precisa de usuario y contraseña, por defecto serán *"anonymous"*. Por lo general este tipo de usuario es el que menos privilegios tiene para manejar los datos del servidor, siendo usuales los permisos de lectura, copiado de datos y listado de carpetas.

6.3.2. Acceso de usuario

Al contrario que el acceso anónimo, este tipo de acceso si precisa de usuario y contraseña y tiene más privilegios. Este tipo de acceso puede acceder a cualquier parte del disco, por lo que solo un administrador se le da este tipo de acceso para que ningún cliente más pueda acceder.

6.3.3. Acceso invitado

Es un acceso hibrido entre el anónimo y el usuario, por una parte requiere un usuario y una contraseña para acceder al servidor, pero por otra no puede acceder a todo el disco en el servidor. Este usuario tiene restringido el acceso a un directorio evitando que puedan acceder a todo el disco, aunque se le puede dar permisos de mayor importancia a la hora de configurar el usuario en el servidor FTP.

6.4. Tipo de transferencia de datos

En el protocolo FTP hay dos tipos de transferencia de datos según los ficheros o información que se transfiere de un equipo a otro.

6.4.1. Tipo ASCII

La transferencia de ficheros es tipo ASCII cuando los ficheros que se utilizan en el traspaso son caracteres imprimibles, como un fichero de texto (.txt) o paginas html.

6.4.2. Tipo Binario

Son los demás ficheros que no son imprimibles por pantalla como por ejemplo ficheros comprimidos, imágenes, videos, etc.

6.5. Seguridad FTP

La seguridad en el protocolo FTP es inexistente, a parte que el servidor puede ser atacado por *spoofing*, también hay que añadir que los datos cuando se transfieren entre cliente y servidor no van cifrados, incluida la autenticación con usuario y contraseña, por lo que pueden ser interceptados desde otro equipo y ser legibles. Se puede hacer que el servidor FTP pueda ser más seguro cambiando el puerto por defecto en el servidor a otro cualquiera mayor que 1024, haciendo accesos de tipo invitados con permisos concretos, pero los agujeros de este protocolo finalmente se verían haciendo posible la intrusión por parte de otros equipos no permitidos.

Por estas razones nacen dos variantes del FTP, el SFTP y el FTPS. El SFTP o también conocido como SSH FTP es un protocolo SSH (*Security SHell*) que está en la capa aplicación al que se le ha añadido el protocolo FTP, mientras que el FTPS es un protocolo FTP que se le ha añadido el protocolo SLL. Tanto el protocolo SSH como el protocolo SSL son dos protocolos de seguridad para la conexión de equipos remotos.

6.5.1. File Transfer Protocol over SSL/TLS

El FTPS es una combinación del protocolo FTP y el protocolo de seguridad SSL/TLS, los datos que vienen del protocolo FTP son tratados después por el protocolo SSL/TLS para cifrar los canales de control/datos a través de comandos que se envían entre cliente y servidor, por lo que este protocolo adquiere las mismas características de seguridad que el protocolo SSL/TLS. A través de estos comandos se define el comportamiento de la conexión y de los algoritmos de cifrado a utilizar mediante la conexión.

Para inicializar la conexión el cliente envía un comando “AUTH” con una petición para establecer una conexión TLS, si el servidor acepta la conexión envía una respuesta 200

aceptando la conexión, de no aceptar el servidor, el cliente puede continuar con la conexión sin seguridad si lo desea. Aunque el cliente pueda continuar con una conexión sin asegurar, el servidor puede rechazar la conexión sin protección de manera que la conexión se cerrará.

Si el servidor ha aceptado iniciar la conexión SSL se crea un canal de conexión para definir los primeros parámetros, para ello pide al cliente que le envié los comandos "USER", "PASS" y "ACCT" para su identificación y para definir los parámetros que identifiquen el tipo de conexión.

Por último el cliente envía el comando "PROT" para establecer el nivel de seguridad que se desea en el canal de datos. El cliente puede enviar dos tipos de protección: *clear*, enviando el comando "PROT c", con el cual se establece un canal inseguro si el servidor lo permite sin protección SSL/TLS, por lo que no hay autenticación, confidencialidad ni integridad entre usuario y servidor. El segundo tipo es *private*, el cliente envía el comando "PROT p" indicando que cliente y servidor tienen que negociar los algoritmos a utilizar dotando al canal de datos la protección deseada.

Una vez que se han negociado los parámetros de seguridad, se inicia el canal de datos que permite la transferencia de ficheros entre servidor y cliente. Durante la vida del canal de datos, el cliente puede enviar el comando "PROT" al servidor indicándole que desea cambiar el nivel de seguridad. Después de enviar el comando "PROT" se envía el comando "REIN", encargado de reinicializar la conexión y obligando a que todos los parámetros sean reseteados, dejando el control de conexión abierto para que el cliente inicie de nuevo el canal de conexión enviando el comando "USER". Si se estuviese transfiriendo algún fichero en ese momento, la transferencia se completaría antes de reiniciar la sesión.

Si durante la sesión con el canal de datos inicializado se requiere que los datos no se transfieran encriptados, el cliente puede enviar al servidor el comando "CCC" (*Clear Command Channel*), para que los datos viajen es texto plano, es decir, sin ninguna protección. El servidor puede rechazar el comando obligando al cliente que envié los datos encriptados, ya que puede suponer un problema de seguridad.

6.5.2. SSH File Transfer Protocol

El SFTP o como se ha nombrado anteriormente, SSH FTP, es un protocolo creado por el grupo IETF (*Internet Engineering Task Force*) para la transferencia de ficheros de manera segura por medio de un protocolo de seguridad, normalmente el SSH. El protocolo SFTP fue diseñado para funcionar sobre un túnel seguro ya creado siguiendo un modelo de petición-respuesta sobre el protocolo SSH2. Aunque se diseñó para realizar la transferencia de ficheros sobre el protocolo SSH2, puede ser usado sobre el protocolo SSL e incluso dentro de una VPN.

Los paquetes generados por este protocolo son tratados por el protocolo SSH2, dotando integridad y confidencialidad a la conexión. La autenticación también es tarea del protocolo SSH2 creando un túnel seguro entre los dos equipos para la transferencia de los paquetes, por lo tanto las características de la seguridad son independientes del protocolo.

Los paquetes generados por este protocolo tienen la siguiente estructura.

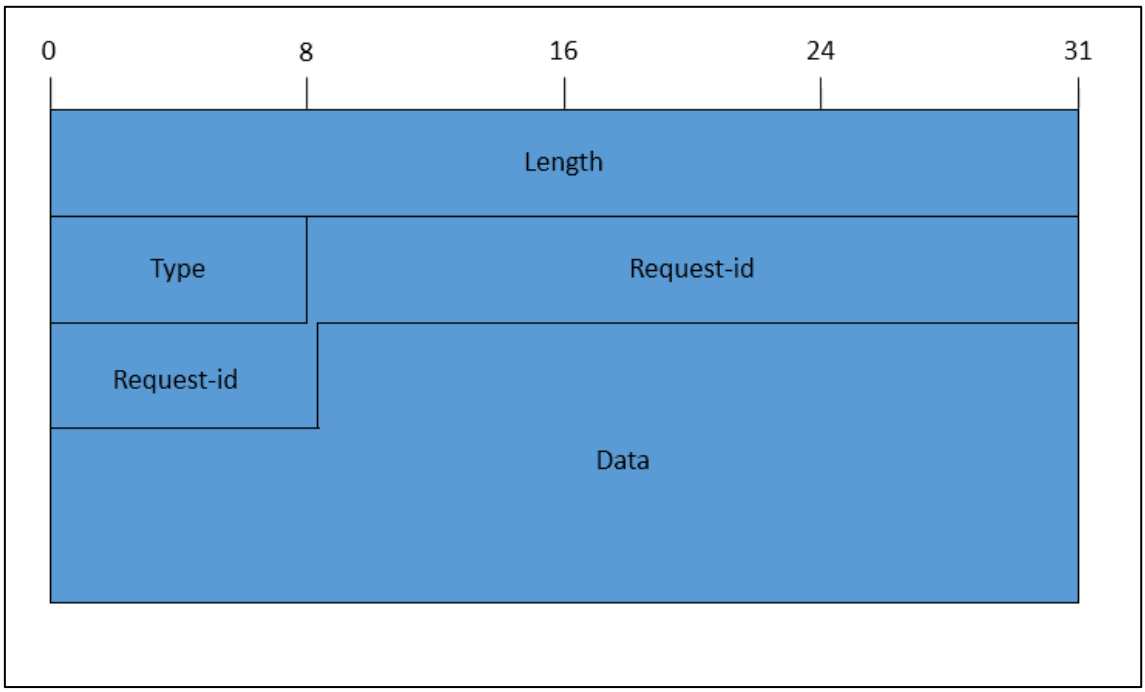


Figura 97. Estructura de un paquete SFTP

- *Length* (4 *bytes*). Indican el tamaño del paquete sin contar con el campo *Length*.

- *Type (1 byte*). Indica el tipo de petición o de respuesta que se está enviando.
- *Request-id (4 bytes).* Es el identificador de la petición, cada vez que el cliente envía un paquete, este campo toma un valor, la respuesta a esta petición por parte del servidor tiene el mismo valor.
- *Data*. El último campo es el de los datos asociados al tipo de mensaje que se envía.

Peticiones del cliente

Cuando el protocolo se inicia, el cliente envía un paquete de tipo *SSH_FXP_INIT* incluyendo la versión que desea usar, el servidor responde con un paquete tipo *SSH_FXP_VERSION* con la versión más baja soportada y la versión enviada por el cliente. Actualmente la última versión es la 6, aunque la más usada por las aplicaciones que hacen uso de este protocolo es la 3.

Los valores que toma el campo *Type* del paquete SFTP con estos dos tipos de inicialización son 1 para *SSH_FXP_INIT* y 2 para *SSH_FXP_VERSION.*

Una vez que se ha acordado que versión del protocolo SFTP a usar, el cliente puede enviar cualquiera mensaje para el manejo y/o manipulación de ficheros siempre y cuando tenga los permisos activados en el servidor. Cada uno de estos tipos tiene una estructura diferente en cuando al campo *Data*. Los valores que toma el campo *Type* van desde el 3 al 23. Las acciones que se pueden realizar son las siguientes:

- Crear, abrir, cerrar, renombrar o borrar un fichero.
- Crear, entrar, renombrar o borrar una carpeta.
- Pedir o modificar un fichero.
- Listar ficheros o carpetas de un directorio.
- Direccionamiento de directorios en el servidor.
- Bloquear o desbloquear un fichero o carpeta que está en uso.

Respuestas del servidor

Todas las peticiones del cliente son respondidas por el servidor enviando un paquete con el tipo *SSH_FXP_STATUS* con valor 101 en el campo *Type* del paquete que se envía.

Si no ha habido ningún error, dentro del paquete *SSH_FXP_STATUS* se envía el estado *SSH_FX_OK* sin datos, sino se envía el tipo de error con los datos asociados a este tipo.

Los paquetes que envía el servidor no siguen un orden, se conoce cual es la respuesta a un paquete de petición porque ambos tienen el mismo *Request-id*.

Otras respuestas que son enviadas por el servidor dependen de la petición que haya hecho el cliente.

- *SSH_FXP_HANDLE*. Indica al cliente el fichero o directorio que el cliente está manipulando. El valor del campo *Type* es 102.
- *SSH_FXP_DATA*. Contiene los datos del fichero que el cliente quiere abrir. El valor del campo *Type* es 103.
- *SSH_FXP_NAME*. Indica el nombre del fichero o directorio al cliente con la petición que ha enviado. El valor del campo *Type* es 104.
- *SSH_FXP_ATTRS*. El servidor envía los atributos de un fichero o directorio del que se ha hecho la petición. El valor del campo *Type* es 105.

Adicionalmente hay dos tipos de paquetes especiales petición-respuesta, el primero es un paquete de tipo *SSH_FXP_EXTENDED* con valor 200 en el campo *Type*, el cliente hace una petición de una información o solicitud que no puede hacer con otro tipo. El servidor puede responder al cliente a este tipo de mensaje con una respuesta vista anteriormente si con ello puede satisfacer la solicitud recibida o con un paquete de tipo *SSH_FXP_EXTENDED_REPLY* con valor 201 en el campo *Type* para enviar la información solicitada por el cliente.

7. DISEÑO DE UNA APLICACIÓN SFTP OPERATIVA EN UNA RED DVB-S/RCS

Como se ha visto en el apartado 5, se ha seguido el modelo de la red **SAICA**. La red **SAICA** es un conjunto de estaciones remotas a lo largo del rio Tajo y una estación central ubicada en Madrid. Cada estación se encarga de recoger muestras de los parámetros de la calidad del agua cada 15 minutos y enviarlas por la red VSAT a la estación central con un hub intermedio. En la siguiente imagen se puede ver el funcionamiento de la red.

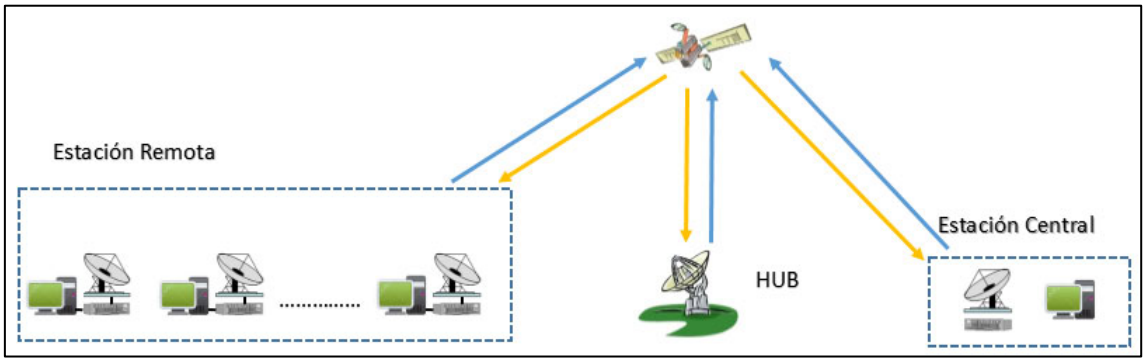


Figura 98. Red VSAT SAICA

El objetivo es la conexión desde la estación central al resto de las estaciones remotas ubicadas a lo largo del rio, en las cuales se instalará un servidor SFTP, para la transferencia de los ficheros de manera segura con los parámetros de las muestras de forma automática que tengan un formato especial.

Para dotar a la red más seguridad de la proporcionada por el estándar DVB-S, se ha pensado en usar una aplicación FTPS o SFTP pero ninguna de las aplicaciones que existen en la actualidad reúnen los requisitos de este tipo de red, la conexión a varias estaciones de forma automática y descarga de ficheros con un formato específico, por lo que se ha optado por las siguientes opciones.

- Creación de una VPN con *IPSec* y diseño de una aplicación FTP.
- Diseño de una aplicación FTPS.

- Diseño de una aplicación SFTP con SSH2.

Cada una de ellas tiene ventajas e inconvenientes, en el caso de la primera opción, se puede crear un túnel a nivel de red con *IPSec* de una forma fácil y segura con las aplicaciones que hay en el mercado, pero no hay ninguna aplicación FTP convencional que haga descargas automáticas en intervalos de tiempo fijados y filtre la transferencia de archivos con un formato especial, por lo que se descarta está opción debido a que se busca eficiencia, y en este caso se tendría que diseñar una aplicación FTP con los requisitos propuestos y además configurar una conexión *IPSec* . La segunda opción y la tercera son similares, protocolos FTP sobre protocolos de seguridad que se establecen entre la capa de aplicación y la de transporte. En este caso se ha optado por diseñar una aplicación SFTP debido a que la seguridad del protocolo SSH2 es mayor a la del TLS 1.2, ofrece cifrados más fuertes, está más orientado a la transferencia de ficheros y el cifrado de los datos se realiza desde el proceso de negociación.

En la actualidad no existe ninguna aplicación SFTP que ofrezca una solución a los requisitos propuestos, todas las aplicaciones que hay en el mercado hacen uso obligatorio de una persona para realizar cada conexión con el servidor, la única solución es el diseño de una aplicación SFTP.

7.1. Lenguaje de la aplicación C#

El lenguaje escogido para el diseño de la aplicación ha sido C#, debido a que es un lenguaje orientado a objetos potente y moderno pero a la vez simple que permite crear todo tipo de aplicaciones que se ejecutan en la plataforma .NET Framework siendo la evolución de los lenguajes C y C++. La plataforma .NET Framework nos ofrece dos componentes importantes, por un lado el CLR (*Common Lenguage Runtime*) que proporciona la ejecución de varios lenguajes como C#, C++ y C a la vez gracias al **código gestionado** que se ejecuta debajo del CLS (*Common Language Specification*) asegurando que todos los lenguajes cumplen una serie de reglas para que las aplicaciones funcionen de un modo uniforme. El segundo componente es la biblioteca

de .NET Framework que cuenta con múltiples clases común a todos los tipos de lenguaje del **código gestionado**.

Para la realización de la aplicación se ha usado el programa Visual Studio 2012. El Visual Studio es un programa que ofrece Microsoft para el diseño de código no solo de C#, también de C y C++. Es un programa muy intuitivo que proporciona ayuda cuando se escribe el código con sugerencias y explicaciones de cada método, clase y evento. Microsoft permite la descarga y uso de este programa de manera gratuita a las universidades, no así a las empresas. Se puede descargar si perteneces a la docencia universitaria en el siguiente enlace:

http://e5.onthehub.com/WebStore/ProductsByMajorVersionList.aspx?ws=f1b11fc4-826f-e011-971f-0030487d8897&vsro=8

7.1.1. Clase SFTP en C#

A la hora de diseñar la aplicación se encontró un problema, en C# no existe una clase nativa que se incluya dentro de las librerías de .NET Framework para realizar conexiones SFTP, sí que existen clases para conexiones FTP como ***FtpWebRequest*** y ***FtpWebResponse*** y conexiones FTPS a partir de .NET 2.0, activando el parámetro ***EnableSsl*** de la clase ***FtpWebRequest***, que permiten hacer conexiones, descarga de archivos, subida de archivos, etc. mediante el protocolo SSL/TLS.

Para solucionar este inconveniente se ha optado por librerías de **código libre** que incluyen clases SFTP como Sharp.SSH y SSH.NET. Entre todas ellas, la más actualizada y con mayor funciones es la librería SSH.NET por lo que se ha optado por implementarla en nuestra aplicación, en cambio el proyecto Sharp.SSH está abandonado dejando sin soporte a muchos usuarios.

Esta librería ha sido diseñada por un grupo de desarrolladores independientes sin ánimo de lucro que ofrecen su proyecto en una plataforma de libre acceso. La última versión estable es la del 7 de abril de 2013, en la actualidad existe una versión beta de la versión del 6 de abril de 2014 pero algunos usuarios de la plataforma están reportando problemas. El proyecto se inspiró en la otra solución Sharp.SSH

reescribiendo todo el código utilizando .NET Framework 4.0 ofreciendo una gran eficiencia en cuanto a ejecución.

Las características de la librería SSH.NET son:

- Ejecución de comandos SSH usando métodos síncronos y asíncronos.
- Comando de retorno que ejecuta el estado de salida y otra información.
- Proporciona la funcionalidad SFTP para operaciones síncronas y asíncronas.
- Proporciona funcionalidades SCP.
- Proporciona un reporte de estado para las operaciones de subida y descarga.
- SFTP permite la precisión de la implementación de la barra de progreso.
- Reenvío de puertos remotos, dinámicos y locales.
- Implementación Shell/Terminal.
- Se puede especificar una frase de paso en la clave de autenticación.
- Uso de ficheros con múltiples claves para la autenticación.
- Soporta los algoritmos de intercambio de claves diffie-hellman-group-exchange-sha256, diffie-hellman-group-exchange-sha1, diffie-hellman-group14-sha1 y diffie-hellman-group1-sha1.
- Soporta los algoritmos de codificación 3DES-cbc, AES128-cbc, AES192-cbc, AES256-cbc, AES128-ctr, AES192-ctr, AES256-ctr, Blowfish-cbc, cast128-cbc, RC4 y Twofish.
- Soporta algoritmos de autenticación de mensaje hmac-md5, hmac-sha1, hmac-ripemd160, hmac-sha2-256, hmac-sha2-256-96, hmac-md5-96 and hmac-sha1-96.
- Soporte claves públicas, contraseñas y teclado interactivo como métodos de autenticación.
- Soporta claves privadas RSA y DSA.
- Soporta algoritmos para la encriptación de clave privada DES-EDE3-CBC, DES-EDE3-CFB, DES-CBC, AES-128-CBC, AES-192-CBC and AES-256-CBC.
- Soporta la autenticación de dos factores o más.
- Soporte para .NET Framework 3.5, Silverlight y Windows Phone.
- Soporte para SOCKS4, SOCKS5 y HTTP Proxy.

Los algoritmos que se van a utilizar para la transferencia de datos son los que recomienda el grupo IETF (*Internet Engineering Task Force*) en el documento RFC4253:

- Algoritmo para la autenticación cliente-servidor: clave privada SSH-RSA.
- Algoritmo para el encriptado de datos: AES128-cbc.
- Algoritmo de compresión: GZIP, es opcional.
- Algoritmo de autenticación del mensaje: HMAC-SHA1-96.

7.2. Estructura de la aplicación

La aplicación está dividida en varias ventanas, la ventana principal se ejecuta cuando se abre la aplicación y las restantes se abren con la interacción del usuario. La ventana principal se muestra en la siguiente imagen.

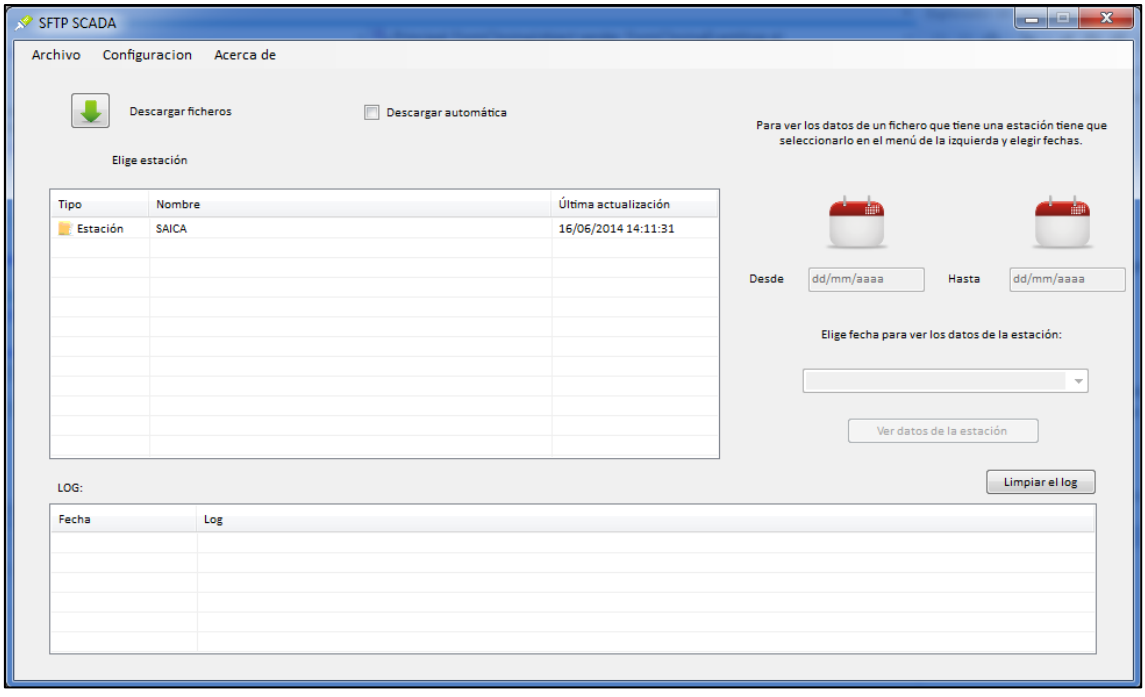


Figura 99. Ventana principal de la aplicación SFTP

1. Ventana principal

En la parte de arriba se encuentra el menú superior de opciones.

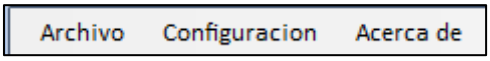


Figura 100. Opciones del menú superior

Debajo de él se encuentran el botón y el *checkbox* para activar la descarga automática.

Figura 101. Opciones de transferencia de datos

En la mitad de la ventana principal aparece una ventana que muestra todas las estaciones como si fuesen carpetas, pudiendo acceder al contenido que hay dentro de ellas al pulsar dos veces con el ratón siempre y cuando se hayan incluido las estaciones en la aplicación previamente.

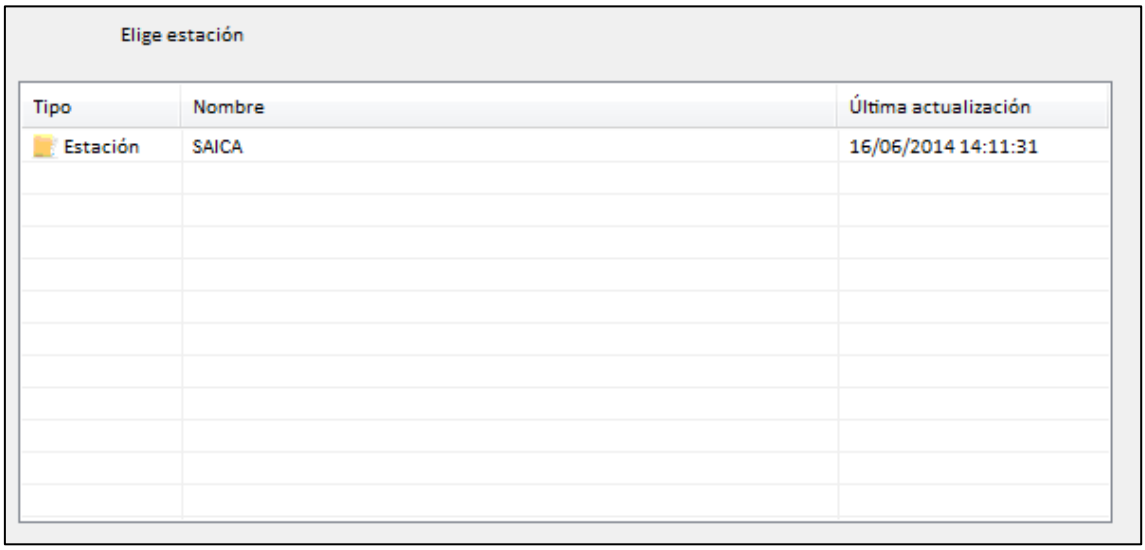


Figura 102. Ventana de estaciones

En la parte derecha está el motor de búsqueda, nos permite hacer la búsqueda de archivos con el formato SAICA entre las fechas indicadas cuando una estación está seleccionada en la ventana de estaciones.

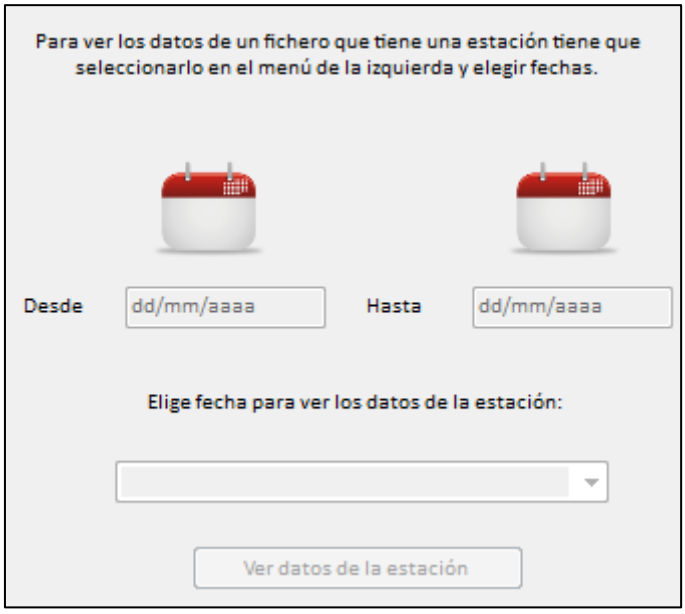


Figura 103. Búsqueda de ficheros

Y por último, la parte de abajo alberga un *log* que recopila todas las acciones que transcurren durante la sesión de la aplicación. Al cerrar la aplicación, toda la información que se ha generado en el *log* se guarda en un archivo de texto en la ubicación que se ha configurado en la ventana de Configuración cuyo nombre es "SFTP SAICA fecha dd-MM-aa hora hh-mm-ss".

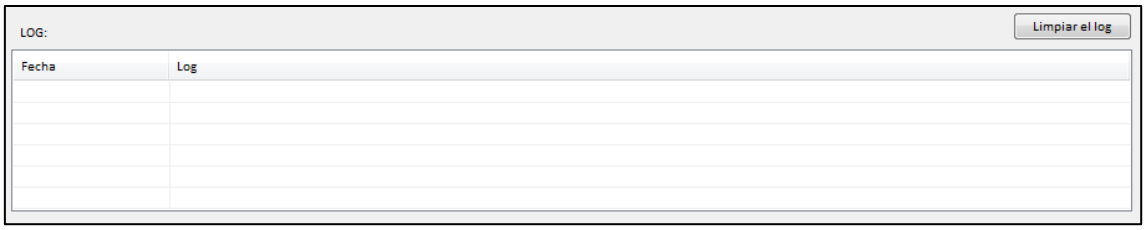


Figura 104. Ventana del log

2. Ventana de Configuración

Para acceder a la ventana de configuración, desde la ventana principal de la aplicación se selecciona la opción de **Configuración** en el menú superior y se abrirá una nueva ventana.

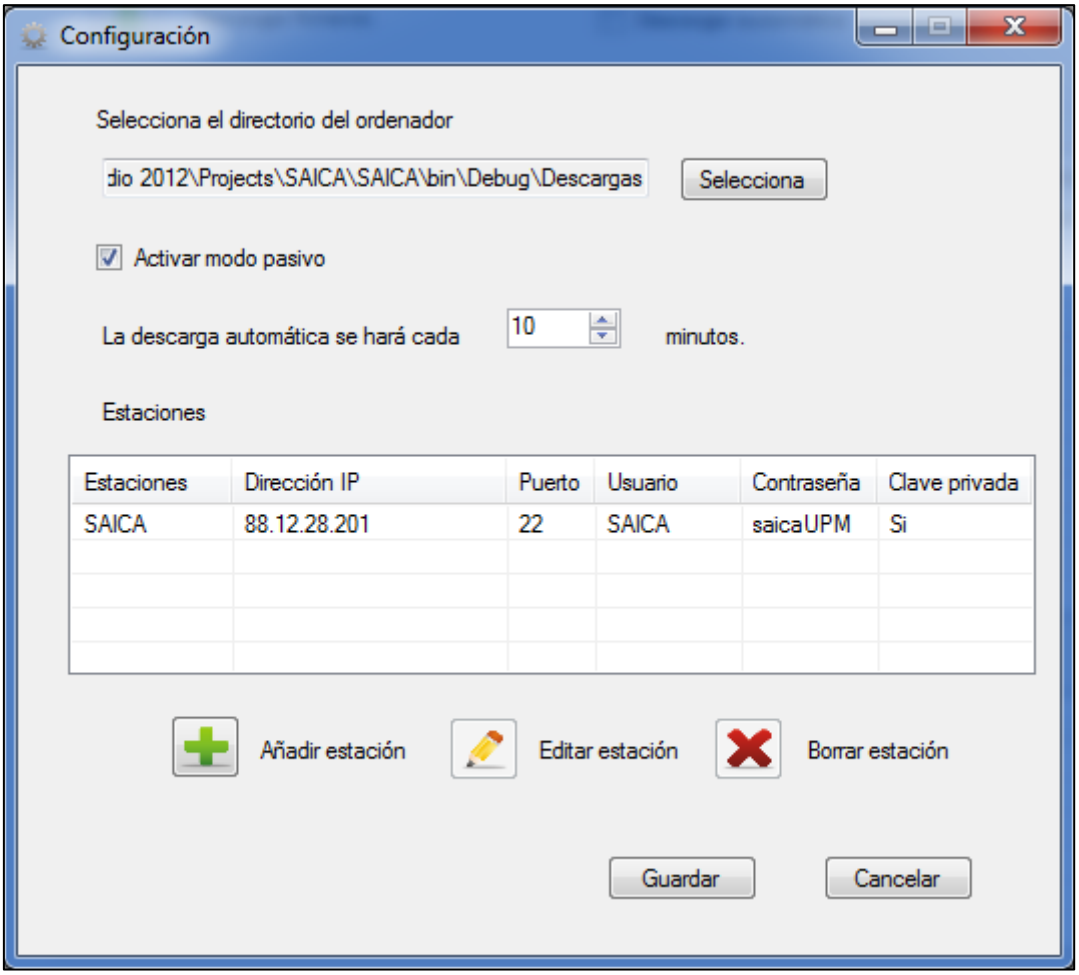


Figura 105. Ventana de Configuración

La ventana de configuración se divide en dos partes de igual tamaño. La parte de arriba tiene tres opciones, selección de la carpeta dónde se guardarán los ficheros de los servidores SFTP, activación del modo pasivo en la conexión y ajuste del intervalo de tiempo para la descarga automática. En la parte de abajo se encuentra las estaciones y la información de cada una de ellas, con las opciones de añadir, editar o borrar una estación.

3. Ventana de nueva/editar estación

Cuando se quiera añadir o editar una estación habrá con pinchar con el ratón los iconos que hay al lado de cada etiqueta, después se abrirá una nueva ventana.

Si se ha pulsado el botón de añadir una nueva estación se abrirá una ventana con todos los campos vacíos.

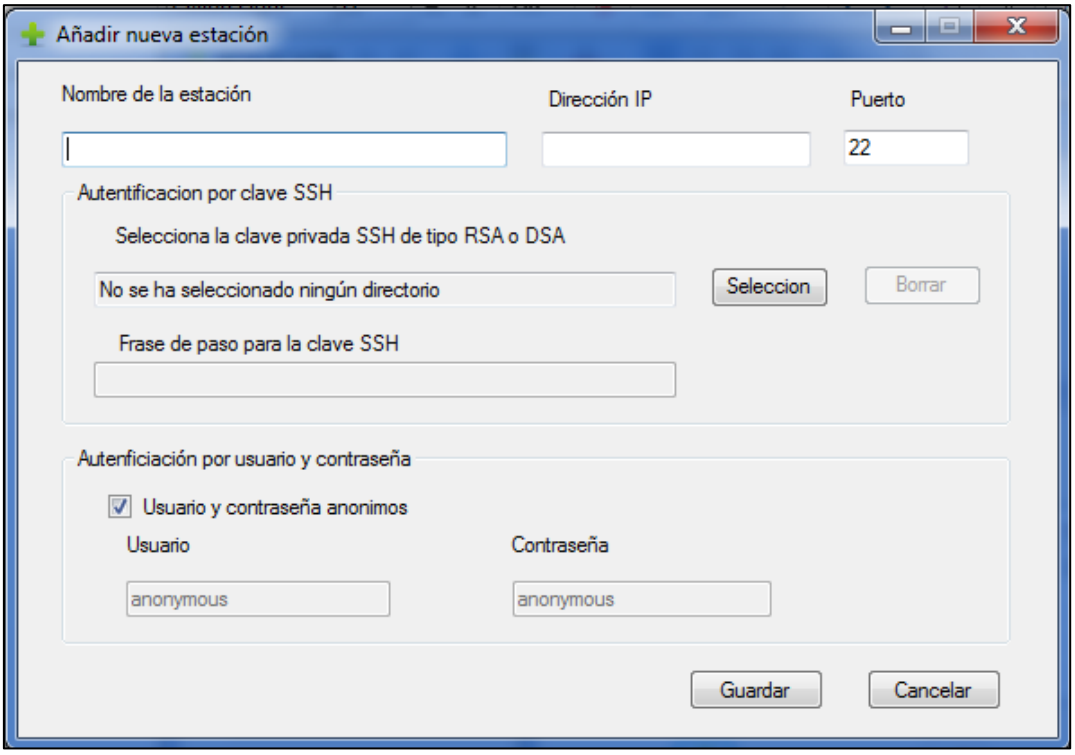


Figura 106. Ventana de nueva estación

Si se ha seleccionado una estación y se ha pulsado el botón de editar estación, se abrirá una ventana con los campos que se guardaron por última vez.

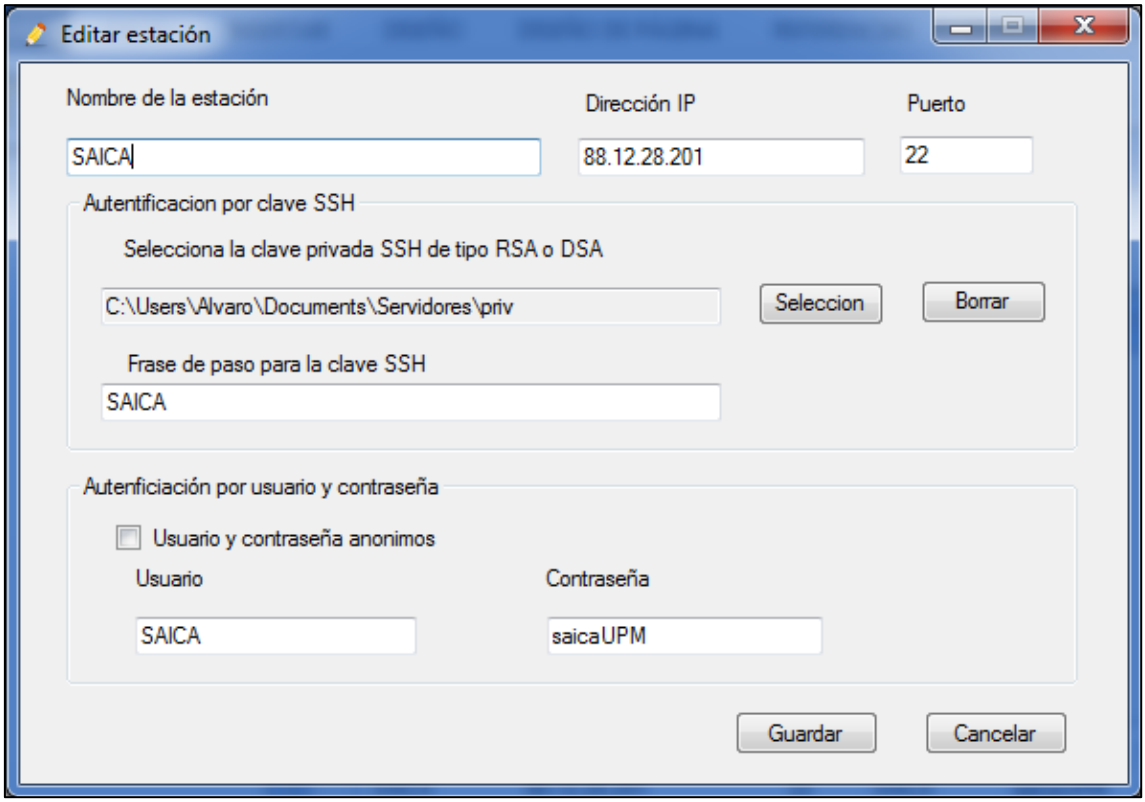


Figura 107. Ventana de editar estación

Al pulsar el botón de **Guardar**, se harán una serie de comprobaciones para asegurarse que los datos son coherentes y si no hay ningún error se guardará la estación y cerrará la ventana.

4. Ventana de datos de una estación

Está ventana se abre de dos maneras, el primer modo es haciendo doble "*click*" con el ratón en un archivo de texto con formato **SAICA** en la ventana principal dentro de una estación/carpeta y el segundo modo es haciendo una búsqueda en la ventana principal, al seleccionar las fechas se incluirán todos los archivos que estén dentro del intervalo y al seleccionar una opción se activará el botón **Ver datos estación,** al pulsarlo abrirá una nueva ventana.

Figura 108. Ventana de datos de una estación

Al pulsar el botón de guardar, se abre una ventana para guardar como texto la información de la trama SAICA de manera legible para cualquier usuario pueda hacer una consulta. Un ejemplo es el siguiente, el fichero que se envía desde una estación del rio Tajo tiene la siguiente información.

0136012C81C2001C000586B033000000000046330000010000 1E33000020002583300003000078

Al guardar los datos desde está ventana, los datos que aparecen dentro del archivo de texto son los siguientes:

Estación Origen: 310
Estación Destino: 300
Clase y tipo de mensaje: 81C2
Longitud de datos: 28
Numero de items: 5
Día Saica: 26/05/2014

} Cabecera trama Saica

QUINCEMINUTO: 51
MEDIOMINUTO: 0
CLASE TIPO: TEMPERATURA
ZONA NUM: 0

} VALOR 1: 70

Primera muestra

QUINCEMINUTO: 51
MEDIOMINUTO: 0
CLASE TIPO: ACIDEZ DEL AGUA
ZONA NUM: 0
VALOR 2: 30

} Segunda muestra

QUINCEMINUTO: 51
MEDIOMINUTO: 0
CLASE TIPO: CONDUCTIVIDAD
ZONA NUM: 0
VALOR 3: 600

} Tercera muestra

QUINCEMINUTO: 51
MEDIOMINUTO: 0
CLASE TIPO: OXIGENO DISUELTO
ZONA NUM: 0
VALOR 4: 120

} Cuarta muestra

5. Ventana AcercaDe

Esta ventana se abre cuando se selecciona la opción **Acerca De** en el menú superior de la ventana principal. Tiene la información de la versión de la aplicación que ahora mismo se encuentra en la 1.0 y los desarrolladores que la han hecho posible.

7.3. Diagramas de la aplicación

En este apartado se va a explicar mediante diagramas las principales acciones de la aplicación para entender el funcionamiento de las conexiones y otros aspectos. El código de la aplicación está al final del proyecto, en él se puede ver todas las líneas que se han diseñado de todas las ventanas vistas anteriormente.

7.3.1. Diagrama conexión estación

Diagrama que conecta una estación, recopila los ficheros que faltan en la estación central y se descargan.

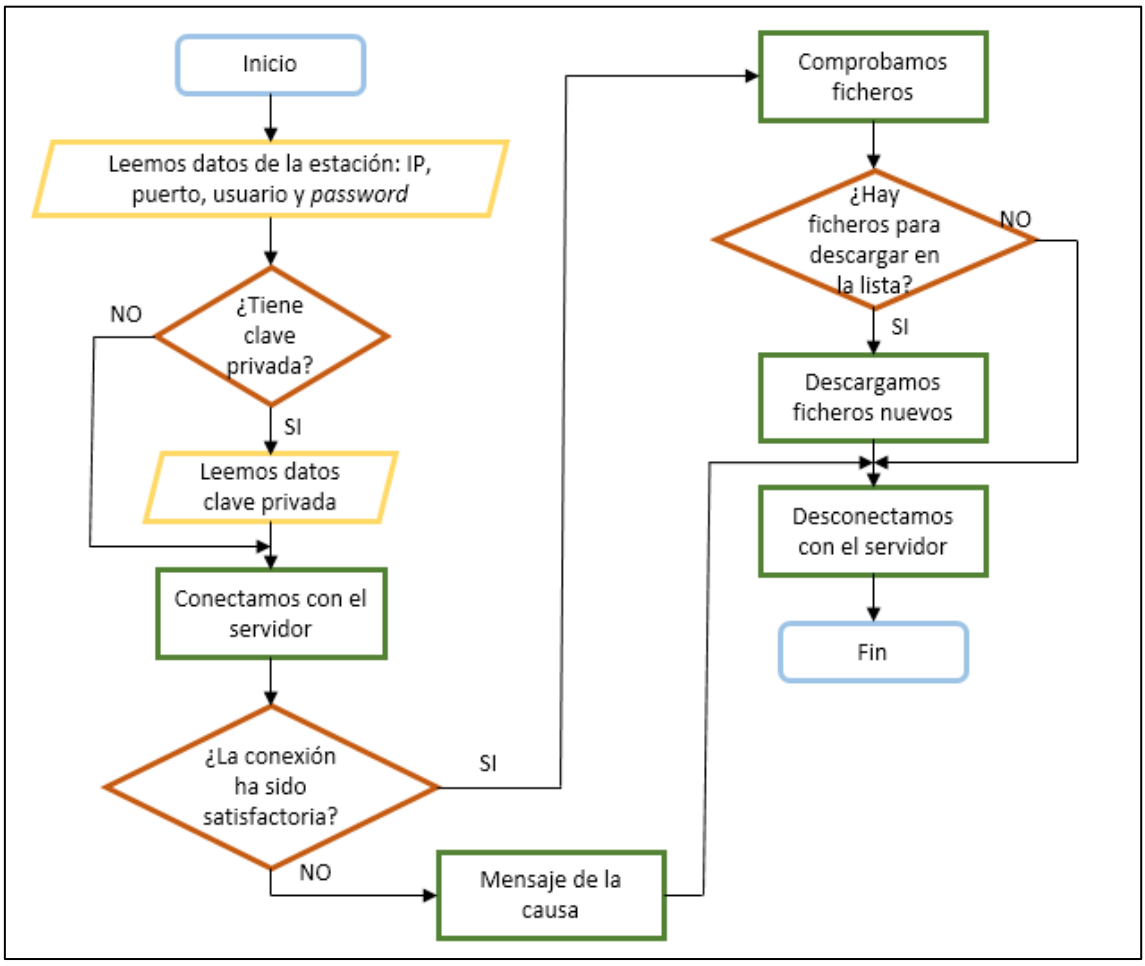


Figura 109. Diagrama de conexión con una estación

7.3.2. Diagrama de comprobación de ficheros

En el diagrama anterior se realizaba una comprobación de archivos que se basa en dos criterios, el primero es comprobar que tiene el nombre de un fichero SAICA.

“EXXX dd-MM-yyyy hh-mm-ss”

Siendo XXX el número de la estación seguido del día, mes, años, hora, minuto y segundo.

El segundo criterio comprueba que en la estación central no se haya descargado el archivo en la estación central.

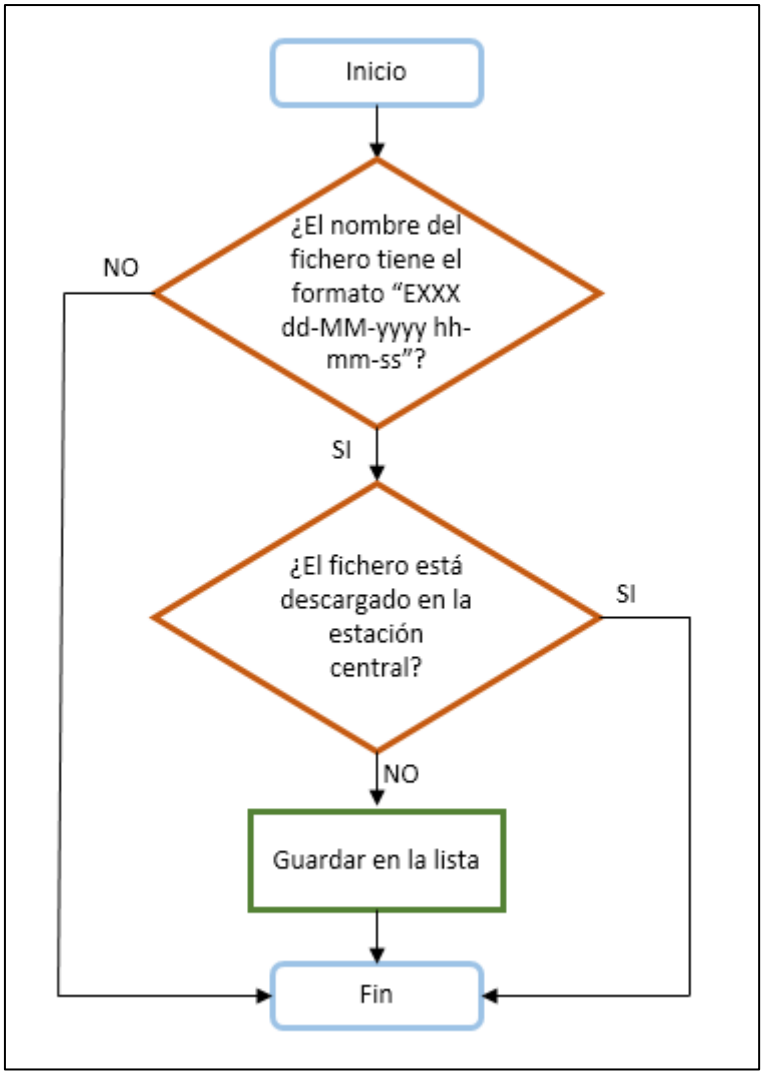


Figura 110. Diagrama de comprobación de ficheros

7.3.3. Diagrama de descarga automática

Al seleccionar el *checkbox* de la ventana principal se activa la descarga automática que se realiza cada intervalo de tiempo seleccionado en la ventana de Configuración.

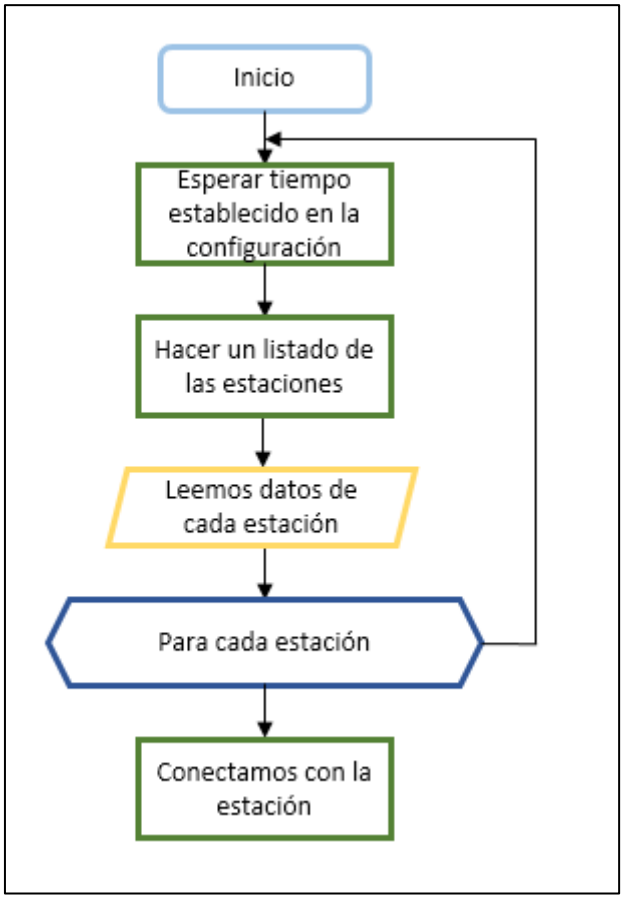


Figura 111. Diagrama de descarga automática

7.3.4. Diagrama de edición o adición de una estación

Al añadir o editar una estación, se comprueban varios campos como el del nombre que no esté repetido, la dirección IP tenga el correcto formato y el puerto sea un número.

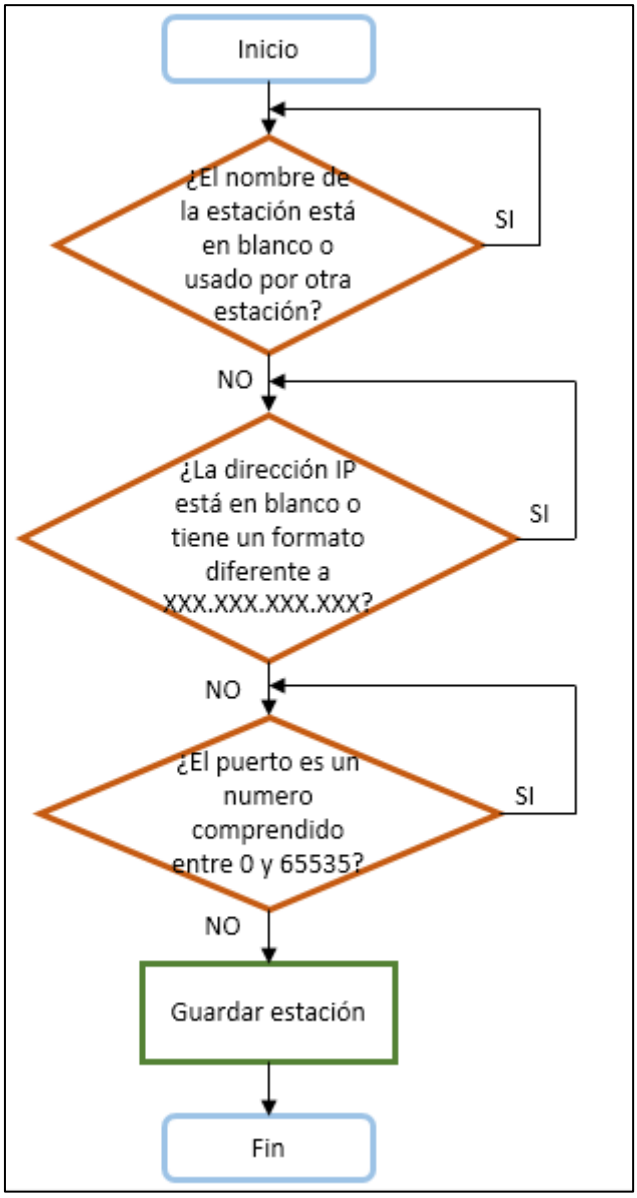


Figura 112. Diagrama de edición o adición de una estación

7.3.5. Diagrama de ver datos de una estación

Para ver los datos de una estación hay dos formas, o hacer doble *"click"* con el ratón en un fichero de texto con el formato **SAICA** que hay dentro de una estación/carpeta situados en la ventana principal o por medio del motor de búsqueda.

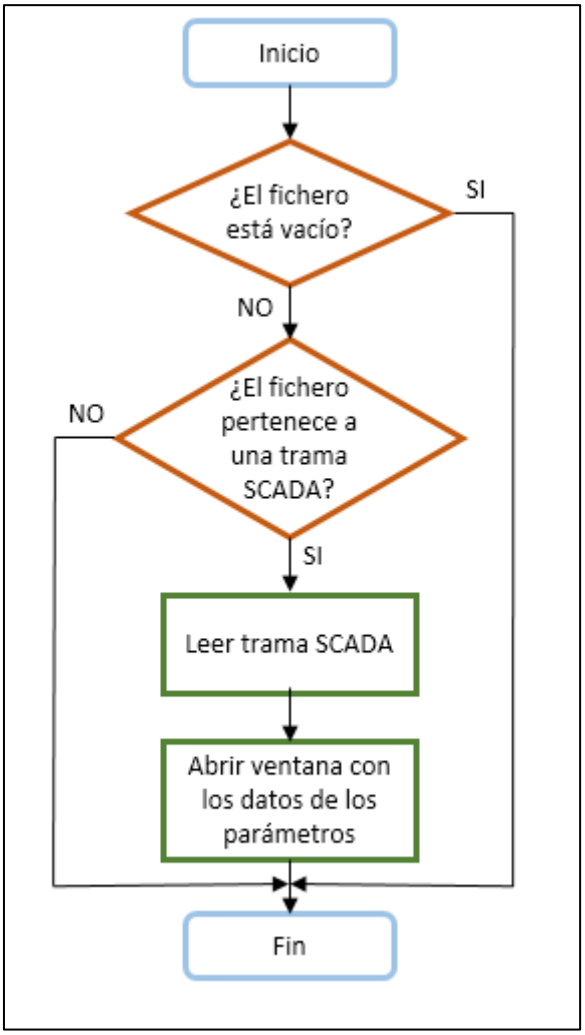


Figura 113. Diagrama para ver los datos de una estación

7.3.6. Diagrama de búsqueda de ficheros

Diagrama de búsqueda que se sitúa en la parte derecha de la ventana principal y nos permite buscar ficheros entre dos fechas.

Figura 114. Diagrama de búsqueda de ficheros

8. PRUEBA DE CAMPO: TRANSFERENCIA DE FICHEROS POR EL PROTOCOLO SFTP EN UNA RED VSAT CON EL ESTANDAR DVB-S/RCS

Gracias a la colaboración con la Conferencia Hidrográfica del Tajo se pudo realizar una prueba de un escenario real de una red VSAT con el estándar DVB-S/RCS, en su red SAICA. La red SAICA explicada en el apartado 5, se compone de varias estaciones remotas VSAT repartidas a lo largo del rio Tajo que recopilan información sobre parámetros importantes de la calidad del agua.

El modelo de la red Saica está basado en una estación central ubicada en Madrid se conecta al resto de estaciones remotas para que estas transfieran los ficheros con los parámetros recopilados. La transferencia de datos se hace mediante un satélite y un hub compartido proporcionado por Telefónica ubicado en la localidad Armuña de Tajuña (Guadalajara) al que no se tiene acceso. El esquema grafico de lo explicado se puede ver en la siguiente imagen.

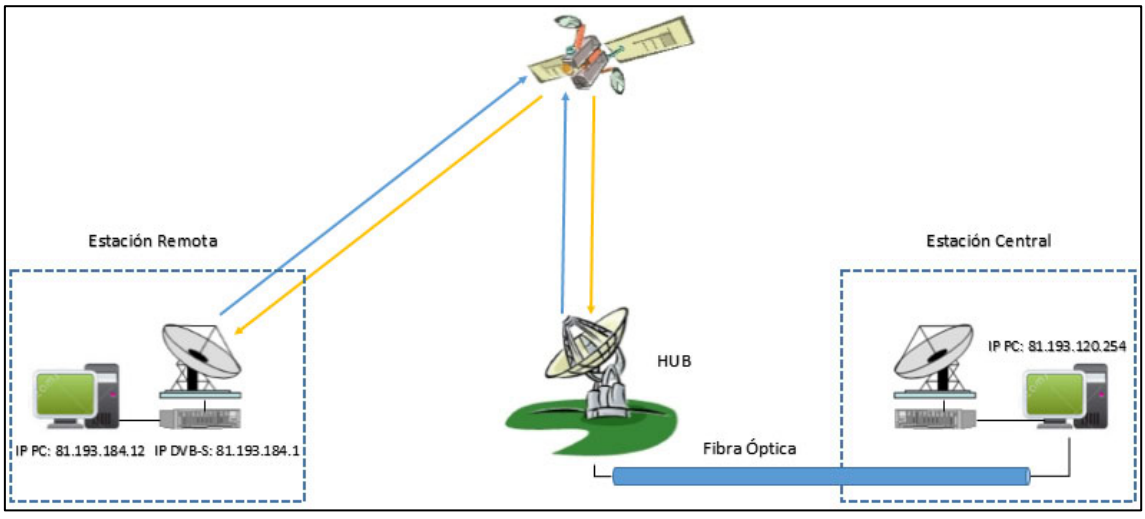


Figura 115. Esquema gráfico de la prueba realizada en la red SAICA

En la visita se tendrá acceso a la estación central y se hará conexión a una estación remota por medio de la red VSAT. Se colocará un servidor de transferencia de ficheros

en una estación remota que será el encargado de generar los ficheros de texto y un cliente en la estación central que pedirá a la estación remota los ficheros de texto con los datos recopilados mediante un cliente de transferencia de ficheros.

8.1. Configuración de los equipos

Lo primero que se hace es ubicar el servidor en una estación remota dentro de la red SAICA y configurar los parámetros IP para acceder a la red, en la estación central se tendrá que hacer el mismo procedimiento.

Para conocer los parámetros IPs de cada estación se ejecuta el comando ***ipconfig*** desde una ventana MSDOS en cada uno de los terminales mostrando por pantalla los siguientes resultados.

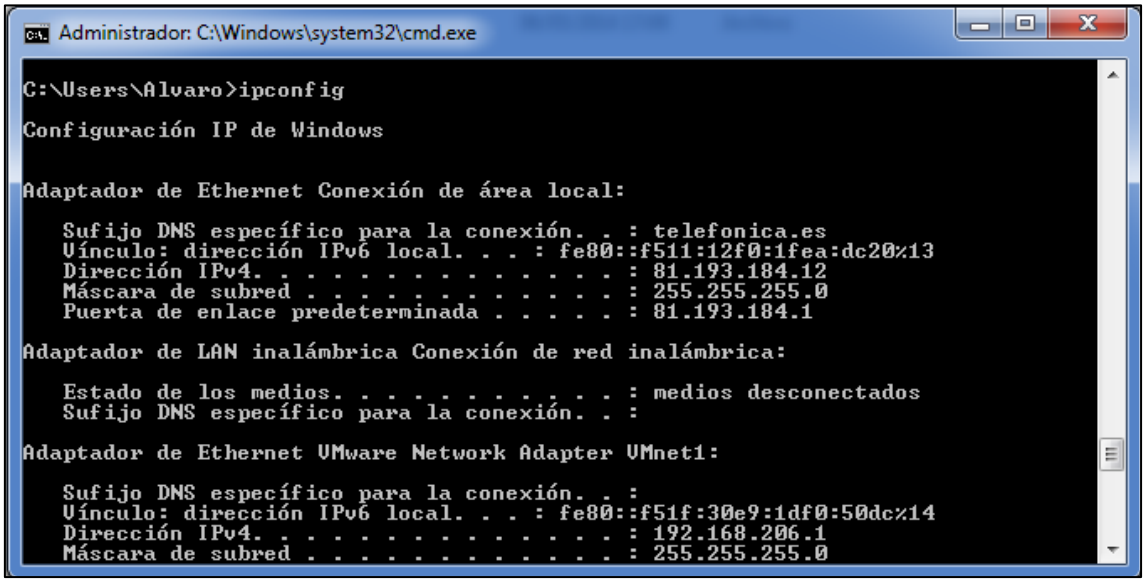


Figura 116. Configuración IP de la estación remota (servidor SFTP)

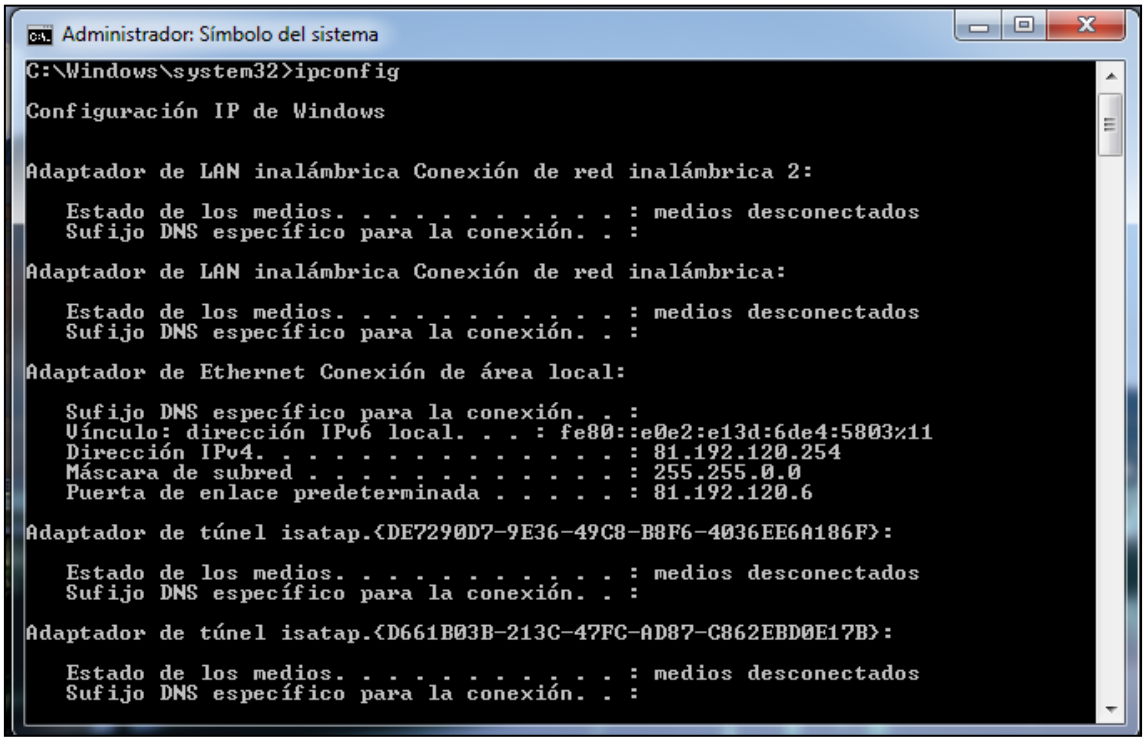


Figura 117. Configuración IP de la estación central (cliente SFTP)

8.2. Prueba de conexión

Para comprobar que ambos dispositivos están conectados a la red y hay conexión entre ellos, es decir, no hay ningún bloqueo por parte de ningún programa y/o enrutador, se hace un ping desde cada uno de los equipos configurados.

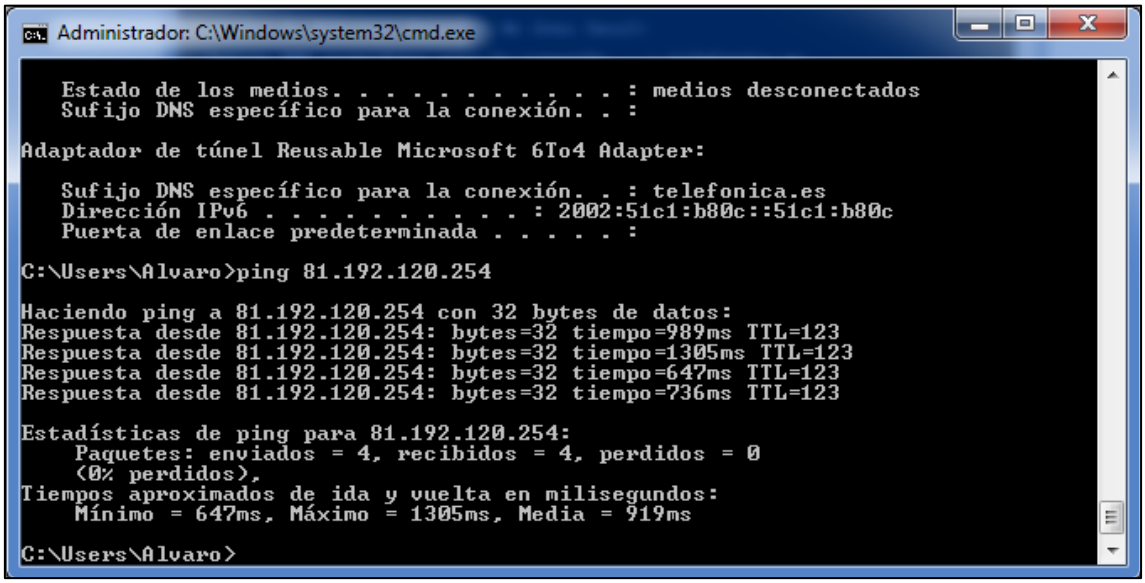


Figura 118. Ping desde la estación remota a la estación central

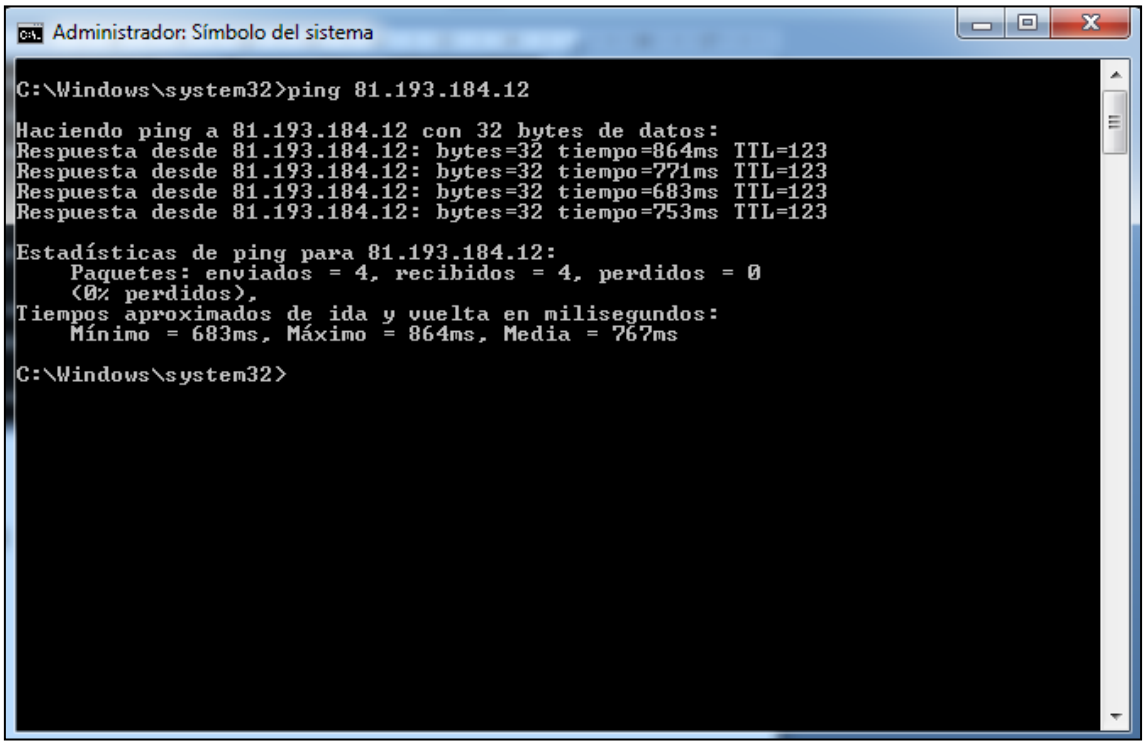


Figura 119. Ping desde la estación central a la estación remota

Como se puede ver en las imágenes hay conexión entre las dos estaciones, obteniendo unos valores de *ping* superiores al que daría si la conexión fuera por una red terrestre. La causa de este retardo es porque la señal tiene que recorrer una mayor distancia, el

satélite empleado para la conexión es un satélite geoestacionario situado a 35000 km sobre la superficie terrestre.

La latencia en una red configurara como la red SAICA sería teóricamente:

$$latencia(sg) = 2 * \frac{distancia(km)}{velocidad(km/sg)} = 2 * \frac{2 \times 35000}{3 \cdot 10^5} = 0.466sg = 466ms$$

En la práctica se puede ver que el tiempo de latencia tiene una media de 466 milisegundos. La diferencia entre la latencia práctica, según la imagen superior de 767ms, y la teórica puede deberse a la suma de diversos factores como el retardo propio de los equipos, la saturación de la red, la posición exacta de la estación con respecto al satélite, etc.

8.3. Prueba de transferencia de datos SFTP

Una vez que se ha comprobado la conexión entre las estaciones se realiza una prueba para la transferencia de ficheros por una red segura. Para ello se instala un servidor SFTP en la estación remota, en este caso el programa usado ha sido el **Core FTP**, se puede encontrar un manual en el Anexo del proyecto que se encuentra al final del mismo. Cuando se ha realizado la instalación y configuración del servidor se procederá a abrir el cliente SFTP diseñado en este proyecto y configurarlo con los parámetros del servidor.

8.3.1. Configuración del servidor SFTP con Core FTP

Lo primero que se debe hacer es conocer las características de seguridad que se quiere que tenga en la transmisión. En la prueba se ha optado por un servidor SFTP con autenticación con claves RSA incluida frase de paso y autenticación con contraseña. Para ello, se inicia la configuración en *Setup->new* con los siguientes parámetros:

- *Domain name:* SAICA

- *Domain IP/Address*: 81.193.184.12
- *Port*: 22
- *Base directory:* C:\Users\Alvaro\Documents\Servidores\SERVIDOR SAICA

Se marcan las opciones *Disable FTP*, *SSH/SFTP*, *Allow key authentication*, *Key authentication only*, *Force password with keys* y *block bounce attacks/FXP*. El resto de parámetros se dejan por defecto o sin checkear.

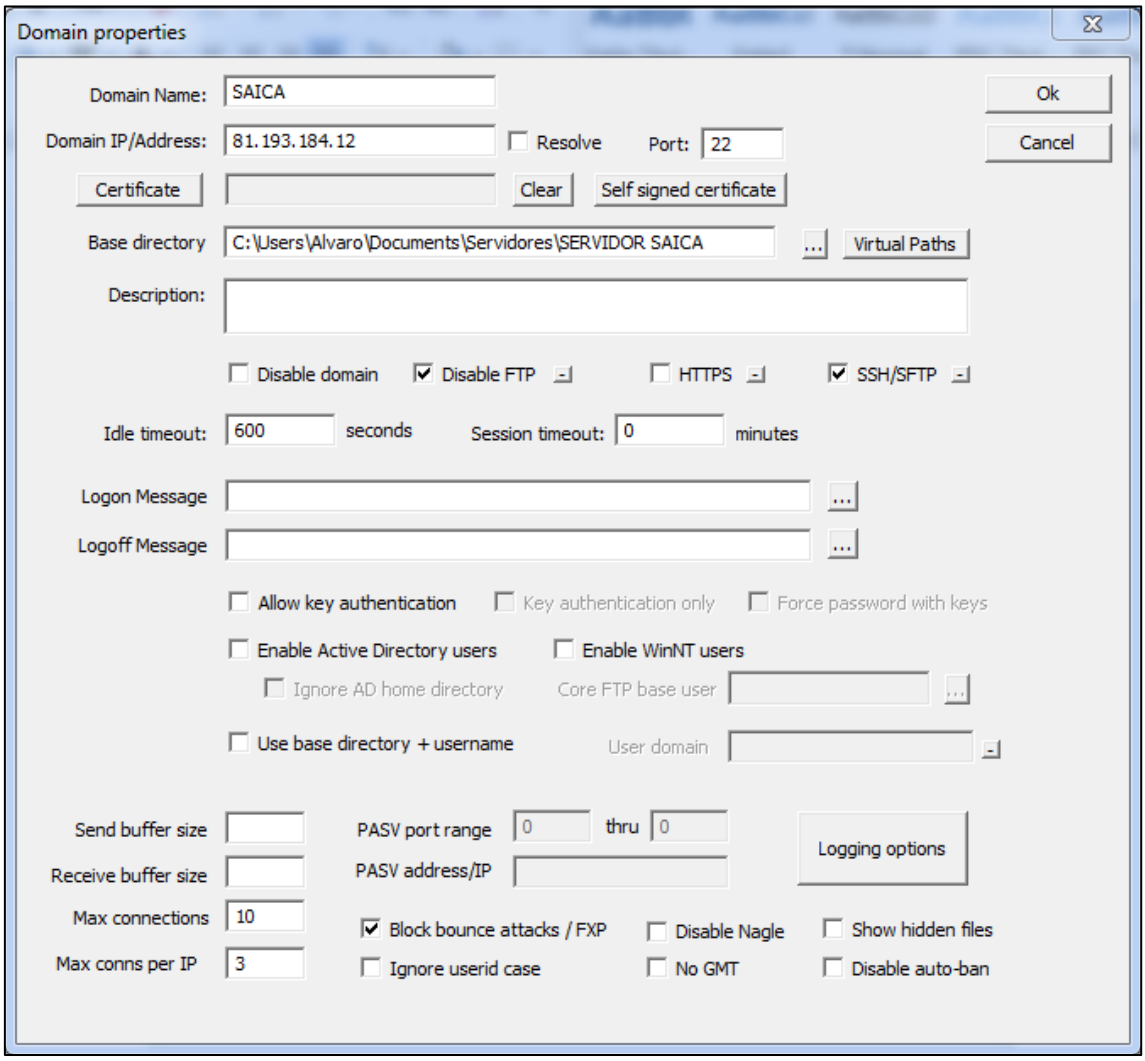


Figura 120. Ventana de creación de un servidor SFTP

Ahora se crea un usuario para el servidor SFTP, para ello se debe seleccionar en el menú *setup* el servidor que se ha creado y después pinchar el botón *new* en el apartado de *users*.

En el apartado General solo se rellena *username* y *password* sin ninguna opción marcada.

- ***Username:*** SAICA
- ***Password:*** saicaUPM

En el apartado Permissions se añade la ruta que se quiere compartir y se activan los siguientes permisos.

- *File Permissions: Read*.
- *Directory Permissions: List.*

Se deja activada la última opción *Inhert rules for subs dir* para que los subdirectorios hereden estos permisos.

Se le da estos permisos, para que el cliente, que será quién recopile los datos solo pueda acceder a los ficheros y carpetas y no pueda modificarlos y/o eliminarlos.

En apartado *Scripts/Commands* se deja por defecto y en el de *Security* se crea un par de claves RSA. Al pulsar el botón *Genere Pair Key* se abre una nueva ventana, se selecciona la ubicación dónde se generarán las claves, el nombre de cada una, en *password* se pone una contraseña también conocida como frase de paso, en nuestro caso **SAICA**, la clave será de 2048 bits, tipo RSA, se marcara la opción *RFC4716 format public key* y por último se pulsara el botón *create*.

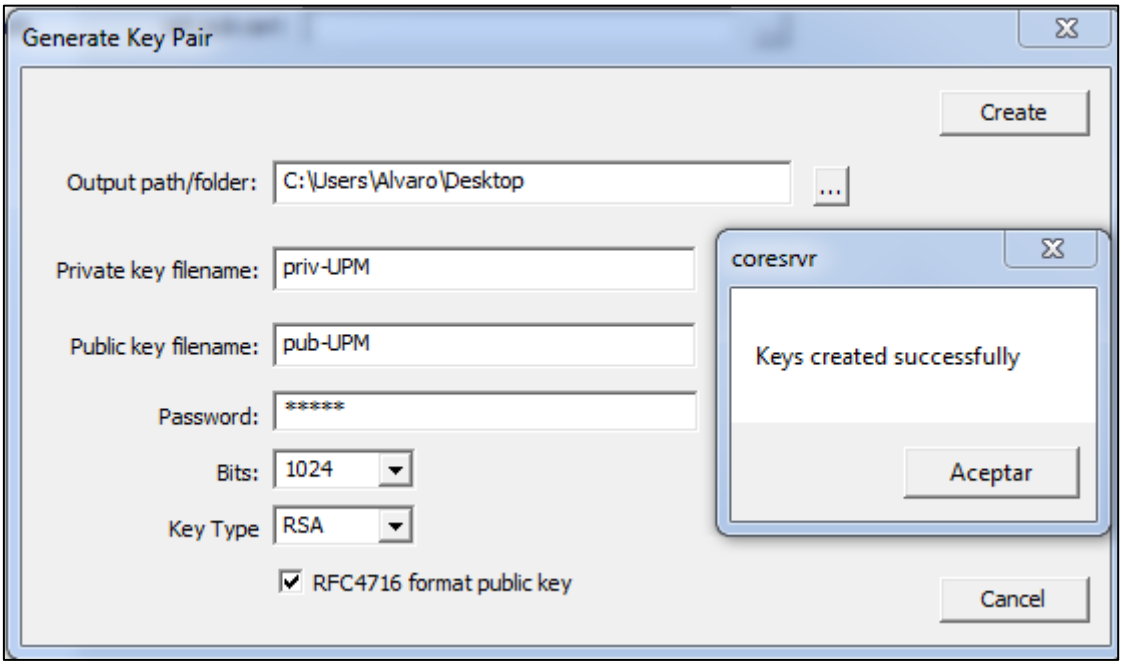


Figura 121. Ventana con la creación de claves privadas

Una vez se genere la clave pública y privada, se añadirá la clave pública en la *opción ssh pub cert* y el servidor ya estará configurado. La clave privada se guardará para incluirla posteriormente en el servidor.

8.3.2. Configuración del cliente SFTP

Se abre la aplicación diseñada en este proyecto y se selecciona la opción del menú superior Configuración. En la ventana de Configuración se configuran los parámetros generales.

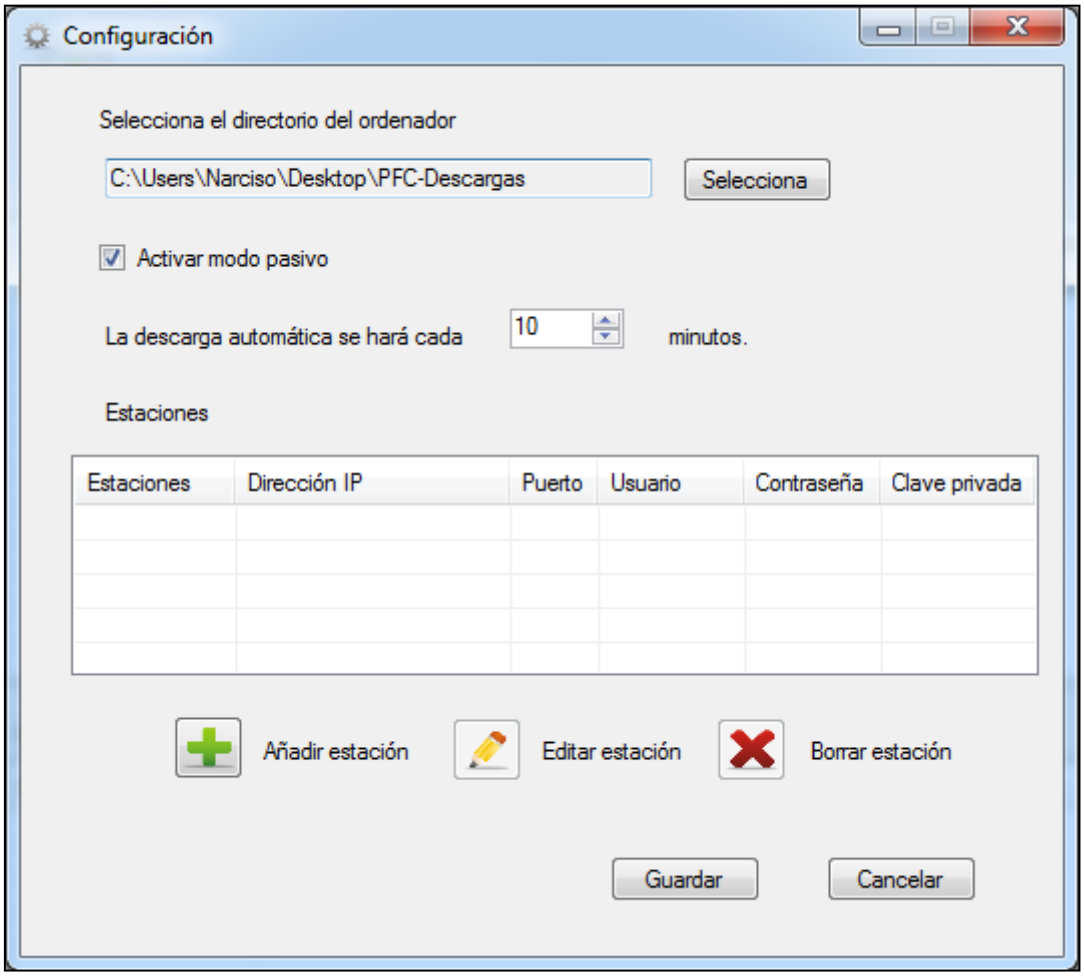


Figura 122. Ventana de configuración del cliente SFTP

Se añade una estación con los datos del servidor SFTP y la clave privada que se ha generado en el servidor y se guarda.

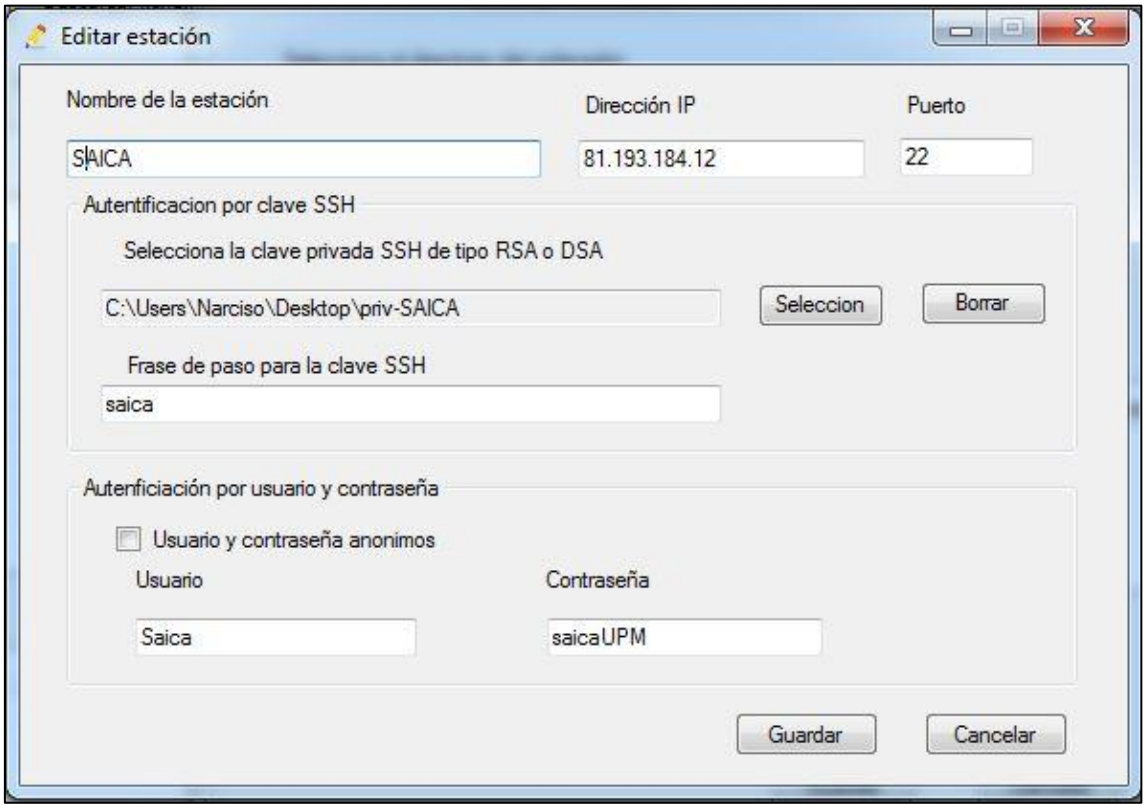


Figura 123. Ventana creación de una estación en el cliente SFTP

La aplicación SFTP está completamente configurada para solicitar los ficheros a la estación remota, el servidor SFTP.

8.3.3. Transferencia de ficheros en la red SAICA

En la ventana principal de la aplicación SFTP se pulsa el botón Descargar ficheros con lo que la conexión hacia el servidor empezará. Todo el proceso se puede comprobar gracias a la ventana situada en la parte inferior que muestra el log de la sesión.

8. PRUEBA DE CAMPO: TRANSFERENCIA DE FICHEROS POR EL PROTOCOLO SFTP EN UNA RED VSAT CON EL ESTANDAR DVB-S/RCS

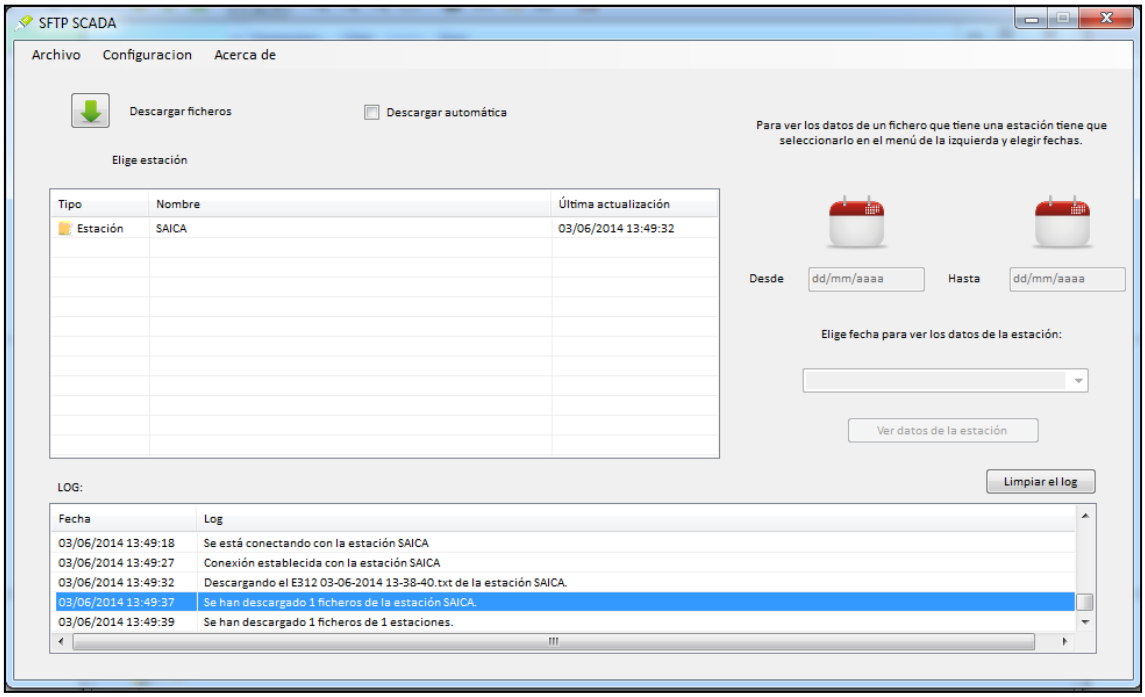


Figura 124. Ventana principal de la transferencia de ficheros.

Una vez que haya terminado la transferencia de ficheros, mediante una ventana se notifica al usuario los ficheros descargados de las estaciones que se ha hecho conexión, en este caso solo de una.

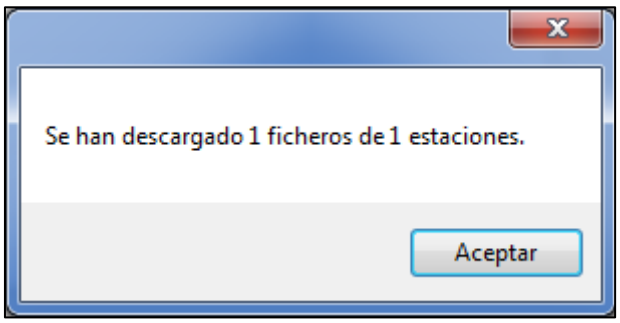


Figura 125. Ventana emergente de finalización de transferencia de datos con las estaciones remotas

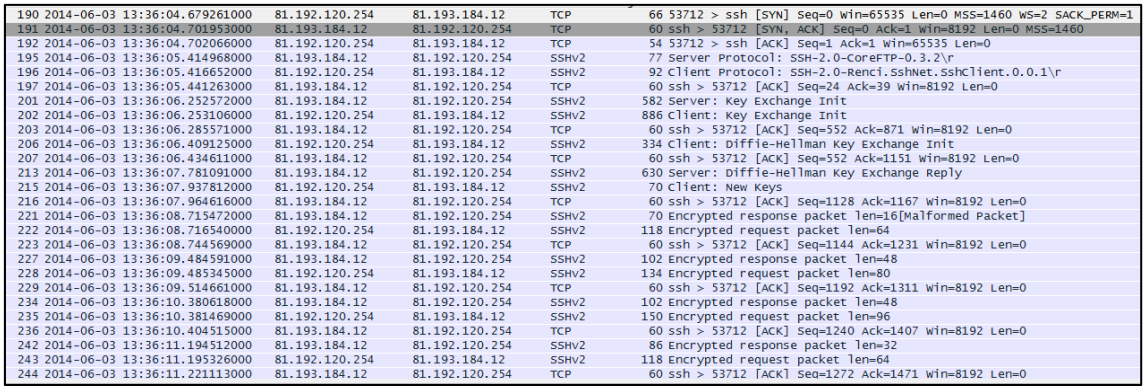
El *log* generado en la sesión ha sido el siguiente:

```
8.  ************************************************************
9.  ***                                                      ***
10. *** LOG SFTP SAICA a fecha 03/06/2014 13:52:30 ***
11. ***                                                      ***
12. ************************************************************
13.
14. 03/06/2014 13:49:18  Se está conectando con la estación SAICA
```

15. 03/06/2014 13:49:27 Conexión establecida con la estación SAICA
16. 03/06/2014 13:49:32 Descargando el E312 03-06-2014 13-38-40.txt de la estación SAICA.
17. 03/06/2014 13:49:37 Se han descargado 1 ficheros de la estación SAICA.
18. 03/06/2014 13:49:39 Se han descargado 1 ficheros de 1 estaciones.

8.3.4. Análisis de paquetes con WireShark

Durante una prueba de transferencia de ficheros se capturó el intercambio de paquetes con el programa **Wireshark**. En la siguiente imagen se puede ver una parte de la captura.



190	2014-06-03 13:36:04.679261000	81.192.120.254	81.193.184.12	TCP	66 53712 > ssh [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=2 SACK_PERM=1
191	2014-06-03 13:36:04.701953000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
192	2014-06-03 13:36:04.702066000	81.192.120.254	81.193.184.12	TCP	54 53712 > ssh [ACK] Seq=1 Ack=1 Win=65535 Len=0
195	2014-06-03 13:36:05.414968000	81.193.184.12	81.192.120.254	SSHv2	77 Server Protocol: SSH-2.0-CoreFTP-0.3.2\r
196	2014-06-03 13:36:05.416652000	81.192.120.254	81.193.184.12	SSHv2	92 Client Protocol: SSH-2.0-Renci.SshNet.SshClient.0.0.1\r
197	2014-06-03 13:36:05.441263000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=24 Ack=39 Win=8192 Len=0
201	2014-06-03 13:36:06.252572000	81.193.184.12	81.192.120.254	SSHv2	582 Server: Key Exchange Init
202	2014-06-03 13:36:06.253106000	81.192.120.254	81.193.184.12	SSHv2	886 Client: Key Exchange Init
203	2014-06-03 13:36:06.285571000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=552 Ack=871 Win=8192 Len=0
206	2014-06-03 13:36:06.409125000	81.192.120.254	81.193.184.12	SSHv2	334 Client: Diffie-Hellman Key Exchange Init
207	2014-06-03 13:36:06.434611000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=552 Ack=1151 Win=8192 Len=0
213	2014-06-03 13:36:07.781091000	81.193.184.12	81.192.120.254	SSHv2	630 Server: Diffie-Hellman Key Exchange Reply
215	2014-06-03 13:36:07.937812000	81.192.120.254	81.193.184.12	SSHv2	70 Client: New Keys
216	2014-06-03 13:36:07.964616000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=1128 Ack=1167 Win=8192 Len=0
221	2014-06-03 13:36:08.715472000	81.193.184.12	81.192.120.254	SSHv2	70 Encrypted response packet len=16[Malformed Packet]
222	2014-06-03 13:36:08.716540000	81.192.120.254	81.193.184.12	SSHv2	118 Encrypted request packet len=64
223	2014-06-03 13:36:08.744569000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=1144 Ack=1231 Win=8192 Len=0
227	2014-06-03 13:36:09.484591000	81.193.184.12	81.192.120.254	SSHv2	102 Encrypted response packet len=48
228	2014-06-03 13:36:09.485345000	81.192.120.254	81.193.184.12	SSHv2	134 Encrypted request packet len=80
229	2014-06-03 13:36:09.514661000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=1192 Ack=1311 Win=8192 Len=0
234	2014-06-03 13:36:10.380618000	81.193.184.12	81.192.120.254	SSHv2	102 Encrypted response packet len=48
235	2014-06-03 13:36:10.381469000	81.192.120.254	81.193.184.12	SSHv2	150 Encrypted request packet len=96
236	2014-06-03 13:36:10.404515000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=1240 Ack=1407 Win=8192 Len=0
242	2014-06-03 13:36:11.194512000	81.193.184.12	81.192.120.254	SSHv2	86 Encrypted response packet len=32
243	2014-06-03 13:36:11.195326000	81.192.120.254	81.193.184.12	SSHv2	118 Encrypted request packet len=64
244	2014-06-03 13:36:11.221113000	81.193.184.12	81.192.120.254	TCP	60 ssh > 53712 [ACK] Seq=1272 Ack=1471 Win=8192 Len=0

Figura 126. Paquetes capturados con Wireshark.

Direcciones IPs y Puertos Utilizados

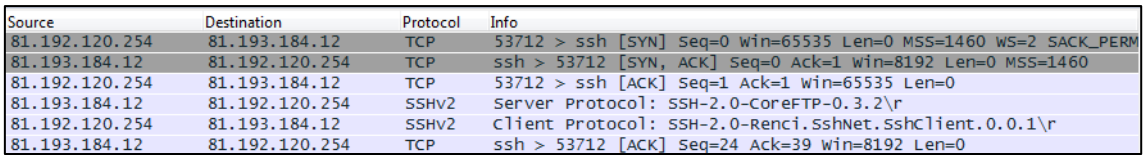
Como se muestra en las líneas de comandos, se puede observar las direcciones IPs y los puertos de cada equipo.

- Dirección IP servidor: 81.193.184.12
- Puerto servidor: 22
- Dirección IP cliente: 81.192.120.254
- Puerto cliente: 53712

En el intercambio de paquetes se pueden diferencia distintas etapas en la conexión haciendo un énfasis en cada una de ellas.

Identificación de protocolos SSH

En los primeros paquetes el cliente hace una petición TCP al servidor para usar el puerto 22 de este último para realizar una conexión SSH. El servidor que está escuchando ese puerto le contesta aceptando la conexión enviando el protocolo SSH, el cliente responde enviando el protocolo SSH a usar.



Source	Destination	Protocol	Info
81.192.120.254	81.193.184.12	TCP	53712 > ssh [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=2 SACK_PERM
81.193.184.12	81.192.120.254	TCP	ssh > 53712 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
81.192.120.254	81.193.184.12	TCP	53712 > ssh [ACK] Seq=1 Ack=1 Win=65535 Len=0
81.193.184.12	81.192.120.254	SSHv2	Server Protocol: SSH-2.0-CoreFTP-0.3.2\r
81.192.120.254	81.193.184.12	SSHv2	Client Protocol: SSH-2.0-Renci.SshNet.SshClient.0.0.1\r
81.193.184.12	81.192.120.254	TCP	ssh > 53712 [ACK] Seq=24 Ack=39 Win=8192 Len=0

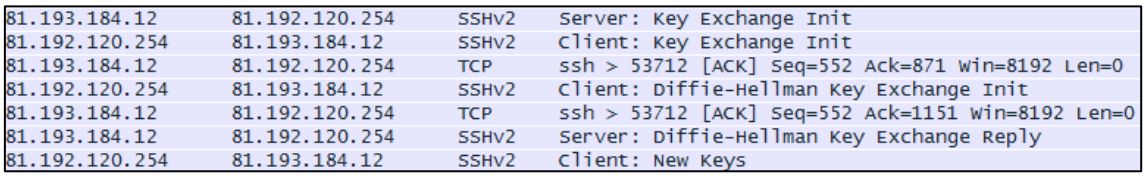
Figura 127. Identificación de los protocolos SSH.

- Protocolo del servidor: SSH-2.0-CoreFtp-0.3.2\r\n

- Protocolo del cliente: SSH-2.0-Renci.SshNetClient.0.0.1\r\n

Intercambio de claves *Diffie-Hellman*

Después el servidor le indica al cliente que se va a realizar el intercambio de claves con los algoritmos soportados, el cliente le responde con los algoritmos que soporta y finalmente el cliente determina las claves que se van a usar en la transmisión.



81.193.184.12	81.192.120.254	SSHv2	Server: Key Exchange Init
81.192.120.254	81.193.184.12	SSHv2	Client: Key Exchange Init
81.193.184.12	81.192.120.254	TCP	ssh > 53712 [ACK] Seq=552 Ack=871 Win=8192 Len=0
81.192.120.254	81.193.184.12	SSHv2	Client: Diffie-Hellman Key Exchange Init
81.193.184.12	81.192.120.254	TCP	ssh > 53712 [ACK] Seq=552 Ack=1151 Win=8192 Len=0
81.193.184.12	81.192.120.254	SSHv2	Server: Diffie-Hellman Key Exchange Reply
81.192.120.254	81.193.184.12	SSHv2	Client: New Keys

Figura 128. Intercambio de claves *Diffie-Hellman*.

Los mensajes intercambiados para la negociación

1. Servidor:
   - ***Message code*:** Key Exchange Init (20)
   - ***Server_host_key_algorithms string*:** ss-rsa
   - ***Encryption_algorithms_server_to_client string*:** aes128-cbc aes128-ctr, 3des-cdc, blowfish-cbc, aes192-cbc, aes192-ctr, aes256-cbc, aes256-ctr, rijndael192-cbc, rijndael256-cbc, rijndael-cbc@lysator.liu.se.

- ***Mac_algorithms_server_to_client string*:** hmac-sha1, hmac-md5, none.

2. Cliente:
   - ***Message code*:** Key Exchange Init (20)
   - ***Encryption_algorithms_client_to_server string*:** aes128-cbc aes128-ctr, 3des-cdc, blowfish-cbc, aes192-cbc, aes192-ctr, aes256-cbc, aes256-ctr, rijndael192-cbc, rijndael256-cbc, rijndael-cbc@lysator.liu.se.
   - ***Mac_algorithms_client_to_server string*:** hmac-sha1, hmac-md5, none.

3. Cliente:
   - ***Message code*:** Diffie-Hellman Key Exchange Init (30)
   - ***Payload*:** 000010100d37685ee4e81c86ef559288512ed7f53b556b0….5ca
   - ***Padding String*:** 9f1882d2909b62af53f3e4dac2

4. Servidor:
   - ***Message code*:** Diffie-Hellman Key Exchange Reply (31)
   - ***DH modulus (P)*:** 000000077373682d727361000000012300000000 8100bcdb39…db4c0ded1acd13 13be8e42e65b604236f24338c3ad.
   - ***DH base (G)*:** 00c0bd5524a389bff74562c225b9cc13e839417228540413c2183d97de33bd…18e 057c50c4f496bc37eb472b0cfce33de84.
   - ***Payload*:** 0000008f000000077373682s727361000000809dd4895571b30cc230ecb2…10cc5 9b83a0d1fc504cddaf476ac3aa4.
   - ***Padding String*:** 000000000000000000

5. Cliente:
   - ***Message code*:** New Keys (21)
   - ***Padding String*:** d676c584144f5448e65

Intercambio de mensajes de datos

A partir de este momento todos los paquetes son encriptados con verificación de mensaje.

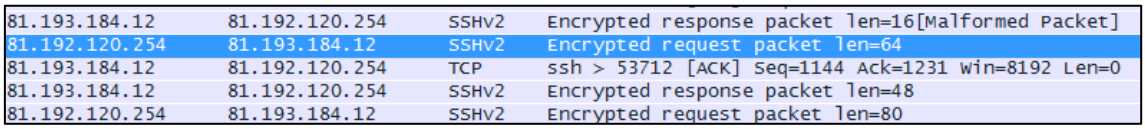


Figura 129. Transferencia de paquetes con encriptación SSH.

Si se abre un paquete encriptado en **Wireshark** y se analiza se puede ver que los paquetes que se están transfiriendo están en todo momento encriptados utilizando un algoritmo de encriptación de datos AES128-cbc y un algoritmo de autenticación de mensaje hmac-sha1.

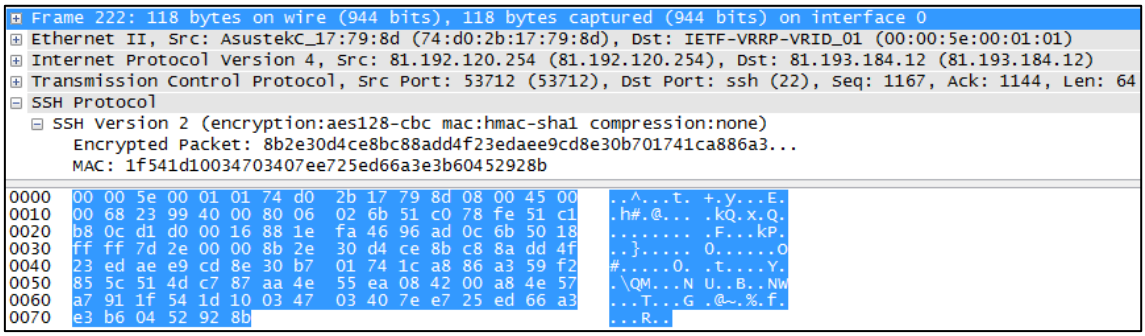


Figura 130. Estructura paquete SSH encriptado.

***Encrypted packet*:**

8b2e30d4ce8bc88add4f23edaee9cd8e30b701741ca886a359f2855c514dc787aa4e55ea084200a84e57a791

MAC: 1f541d10034703407ee725ed66a3e3b60452928b

Fin de la conexión

Cuando la transferencia de archivos ha terminado el servidor cierra la conexión sin avisar al cliente, este envía un paquete *reset* TCP intentando recuperar la conexión.

Figura 131. Paquete TCP con la acción *reset*.

9. CONCLUSIONES

En este capítulo se presentan las conclusiones obtenidas del estudio de este Proyecto Fin de Carrera. Las conclusiones se pueden dividir en tres partes diferenciadas.

En primer lugar se ha visto como las redes VSAT son una magnifica solución cuando se quiere conectar estaciones que están muy dispersas entre si y en donde el difícil acceso a las mismas o la poca infraestructura terrestres del lugar hacen imposible una conexión por los medios convencionales. Además, gracias a su flexibilidad se pueden eliminar o añadir estaciones fácilmente según las necesidades del propietario de la red. El ejemplo práctico que se ha visto en el proyecto es el de la red SAICA, creada por la Confederación Hidrográfica del Tajo para controlar la calidad del agua del río Tajo, en donde existen un número elevado de estaciones distribuidas a lo largo del cauce del río con difícil acceso.

En segundo lugar, se ha mencionado el inconveniente de utilizar un estándar propietario para el funcionamiento de este tipo de redes y las ventajas de utilizar un estándar abierto, que facilita la interoperabilidad entre equipos de diferentes fabricantes. En el caso de este proyecto se ha estudiado la opción de DVB-S por ser uno de los estándares más aceptados internacionalmente y concretamente por ser el empleado en la red SAICA. Este es un estándar flexible que permite tanto la transmisión de video y audio codificado como la transmisión de datos en bruto a través del flujo de transporte, permitiendo un gran abanico de servicios.

Por último, en tercer lugar, se ha comprobado que la implantación de seguridad a través de las capas superiores a los datos enviados se puede hacer de una forma eficaz y barata, tan solo siendo necesaria la existencia de un programa que permita el uso de algoritmos de cifrado para que los datos se transfieran de un modo seguro.

También ha sido posible implantar varios algoritmos de autenticación al realizar la conexión, tanto en la capa del DVB-S como en el subnivel que se crea en el nivel TCP, obligando a la estación remota y la estación central a autenticarse entre ellas mediante

algoritmos de envió de claves simétricas, siendo cada autenticación independiente y por lo tanto dando una mayor seguridad.

Aunque la opción escogida para asegurar la transferencia de ficheros ha sido el SSH, se podían haber usado cualquiera de las otras dos, dado que las características de los tres protocolos cuentan con fuertes mecanismos de protección contra ataques de todo tipo.

Para concluir el proyecto se han creado dos aplicaciones, la primera como método didáctico para comprender mejor el comportamiento de las redes VSAT con el estándar DVB-S, y una segunda aplicación con carácter comercial para la transferencia de ficheros de manera segura con características específicas, enfocada particularmente en redes VSAT, aunque siendo posible su uso en otras redes.

10. FUTURAS LÍNEAS DE TRABAJO

A continuación se presentan varias sugerencias para futuros trabajos relacionados con este proyecto.

El proyecto se ha centrado en el estudio de los estándares de transmisión DVB-S y DVB-RCS por ser los más extendidos actualmente. No obstante, existen las nuevas versiones de los mismos que ofrecen una mayor eficiencia en el uso de los recursos así como una gama más amplia de aplicaciones. Respecto a este asunto se proponen dos futuros trabajos:

- Estudiar las nuevas versiones de los estándares, el DVB-S2 y el DVB-RCS2, comparándolos con las presentadas en el proyecto.
- Evaluar los costes que supondría hacer una migración de los sistemas actuales que trabajan con DVB-S y DVB-RCS a sus nuevas versiones.

Por otro lado se ha visto como uno de los principales problemas de emplear redes satelitales es el **doble salto**, así que un futuro trabajo podría ser estudiar la posibilidad de integrar el hub en el propio satélite. En este proyecto solo se han presentado algunas soluciones al caso concreto de TCP sin entrar en demasiados detalles y asumiendo como inevitable ese **doble salto**.

También se ha comprobado que la seguridad en las redes hoy en día es robusta debido a que los equipos que hay en la actualidad tardarían miles de años es descifrar los algoritmos que utilizan los protocolos de seguridad que hay, pero con el avance tan rápido de la tecnología, el tiempo que tardaría los equipos disminuiría de forma exponencial. Debido a este problema que puede presentarse en un futuro se proponen dos opciones para hacerle frente:

- Aumentar los bits de los algoritmos todas las claves que intervienen en el proceso de autenticación, encriptación y verificación de cada paquete.
- Utilizar dos claves, ya sean simétricas o de tipo pública-privada en cada proceso de la conexión al igual que en el proceso de encriptación de datos antes de ser enviados a los niveles inferiores.

Las dos opciones propuestas harían disminuir la cantidad de bits de datos que se envían por la red, esto significa que se quiere transmitir la misma información, con estas dos opciones se enviarán más paquetes. Aunque puede ser un problema porque se haría un uso de la red menos eficiente, hay que considerar que las técnicas de modulación han mejorado y la transferencia de bit por el mismo espectro de frecuencia ha aumentado con lo que el ancho de banda es mayor y evolucionará con el paso de los años.

Por último se podría implantar un sistema de imagen y sonido en cada estación remota de la red que se ha estudiado, siendo ejemplo para otras muchas, que se muestre en la aplicación de cliente SFTP diseñada en este proyecto y monitorizar los parámetros de la calidad del agua en tiempo real. Con estas mejores se podría hacer un seguimiento más exhaustivo.

11. BIBLIOGRAFÍA

[1] Fischer, Walter; "*Tecnologías para la Radiodifusión Digital de Video y Audio*". Segunda edición, 2008. Berlín: Springer-Verlag.

[2] Maral, Gérard; *"VSAT Networks"* Segunda edición, 2003. Toulouse: Wiley.

[3] ISO/IEC 13818-1 *"Coding of moving pictures and associated audio – Part 1: Systems"*.

[4] ETSI TR 101 202 *"Digital Video Broadcasting (DVB); Implementation guidelines for Data Broadcasting"*.

[5] ETSI EN 300 421 *"Digital Video Broadcasting (DVB); framing structure, channel coding and modulation for 11/12 GHz satellite service"*.

[6] ETSI EN 300 468 *"Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems"*.

[7] ETSI EN 301 192 "*Digital Video Broadcasting (DVB); DVB specification for data broadcasting"*.

[8] ETSI EN 301 790 *"Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems"*.

[9] "*Digital Video Broadcasting (DVB), Return Channel via Satellite (DVBRCS) Background Book*". *Nera Broadband Satellite AS*, November *2002*.

[10] Delgado, Alejandro; "*Flujos de Programa y de Transporte MPEG-2, aplicación a DVB*". Universidad Politécnica de Madrid, Junio 2001.

[11] Barba Molina, Hernán; Chafla Altamirano, Juan; "*Simulación de una red VSAT Full-Duplex para acceso a Internet usando la plataforma DVB-S y DVB-RCS*". Escuela Politécnica Nacional, Diciembre 2006. http://bibdigital.epn.edu.ec/handle/15000/9860

[12] Cruickshank, H; Howarth, M.P; Iyengar, S; Sun, Z; "*A Comparison between satellite DVB conditional access and secure IP multicast"*. University of Surrey, June 2005.

[13] Cruickshank, H; Howarth, M.P; Iyengar, S; Sun, Z; Claverotte, L; "*Security multicast in DVB-RCS satellite systems*". IEEE Wireless Communications, October 2005.

[14] Collini-Nocker, Bernhard; Fairhurst, Godred; "*ULE versus MPE as an IP over DVB Encapsulation*".

[15] Página oficial del DVB Project
http://www.dvb.org

[16] Müller, Luis; *"Seguridad en la capa de Red. – IPSec"*. Universidad Americana, 2011.

[17] IETF RFC 6101 *"The Secure Sockets Layer (SSL) Protocol Version 3.0".*

[18] IETF RFC 5246 *"The Transport Layer Security (TLS) Protocol Version 1.2."*.

[19] IETF RFC 4251 *"The Secure Shell (SSH) Protocol Architecture".*

[20] IETF RFC 4252 *"The Secure Shell (SSH) Authentication Protocol".*

[21] IETF RFC 4253 *"The Secure Shell (SSH) Transport Layer Protocol".*

[22] IETF RFC 4254 *"The Secure Shell (SSH) Connection Protocol"*.

[23] Barrett, D. J.; Silverman, R.; *"SSH, The Secure Shell: The Definitive Guide."* 2001.

[24] IETF RFC 959 *"FILE TRANSFER PROTOCOL (FTP)".*

[25] IETF RFC 4217 *"Securing FTP with TLS"*.

[26] IETF RFC *"SSH File Transfer Protocol".*

[27] Servidor Core FTP/SFTP
http://www.coreftp.com/server/index.html

[28] Librería clase SFTP para C#
https://sshnet.codeplex.com/

[29] Librerias Microsoft .NET
http://msdn.microsoft.com/library/

[30] Foro de programación http://stackoverflow.com/

ANEXO

MANUAL SERVIDOR SFTP CORE FTP

Para realizar el montaje de una red segura por VSAT con DVB-S/RCS que siga una arquitectura cliente-servidor como la planteada en este proyecto o la que sigue la red SAICA, hay que configurar un servidor SFTP en cada estación de la que se desee recopilar información y un cliente SFTP en la estación donde se quiera compilar toda esa información.

Para el servidor se ha escogido el programa **Core FTP** por ser un software gratuito que tiene la ventaja de ser fácilmente configurable, pero en donde la interfaz no es muy intuitiva. Este software permite el uso de los siguientes protocolos: FTP, SSL/TLS v2&3, SSH2/SFTP, HTTPS. El primer paso que se debe hacer para su puesta en funcionamiento es la descarga del mismo a través de la página web oficial del software (**http://www.coreftp.com/server/index.html**), pudiendo escoger la versión de 32 bits o 64 bits, dependiendo de la arquitectura y sistema operativo que tenga el equipo. Después de la descarga se procede a la instalación del mismo ejecutando el ejecutable "*.exe*" descargado.

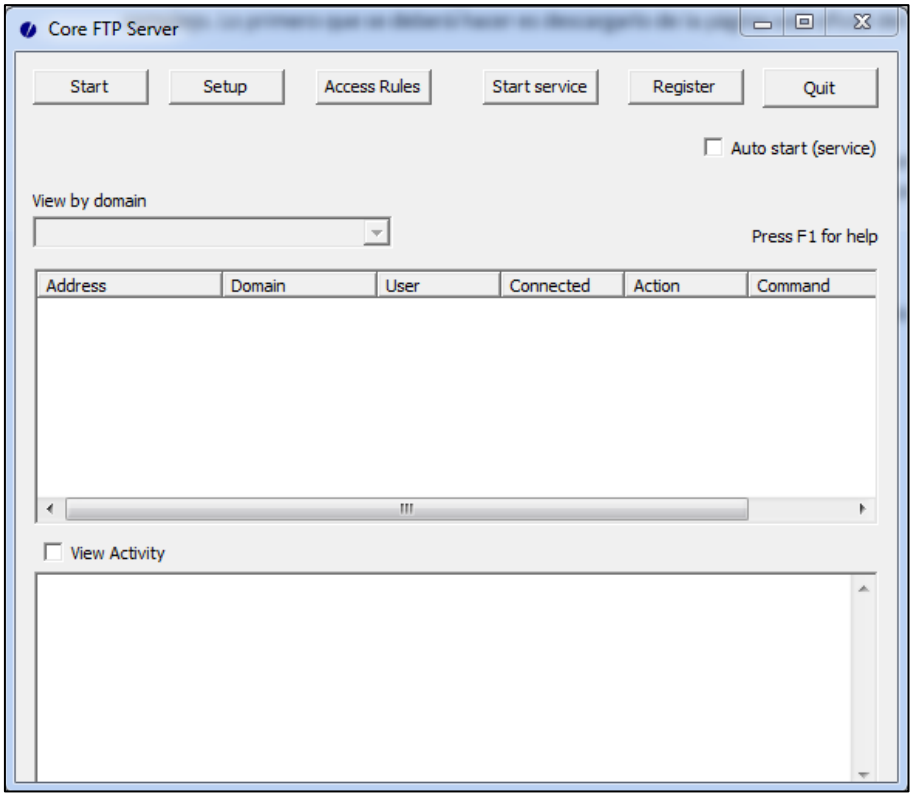


Figura 132. Ventana principal del servidor Core FTP.

La imagen superior muestra la ventana principal del programa. La primera vez que se abre la aplicación se tiene que pulsar el botón *Setup* para abrir el submenú de configuración, apareciendo una ventana como la que se muestra a continuación.

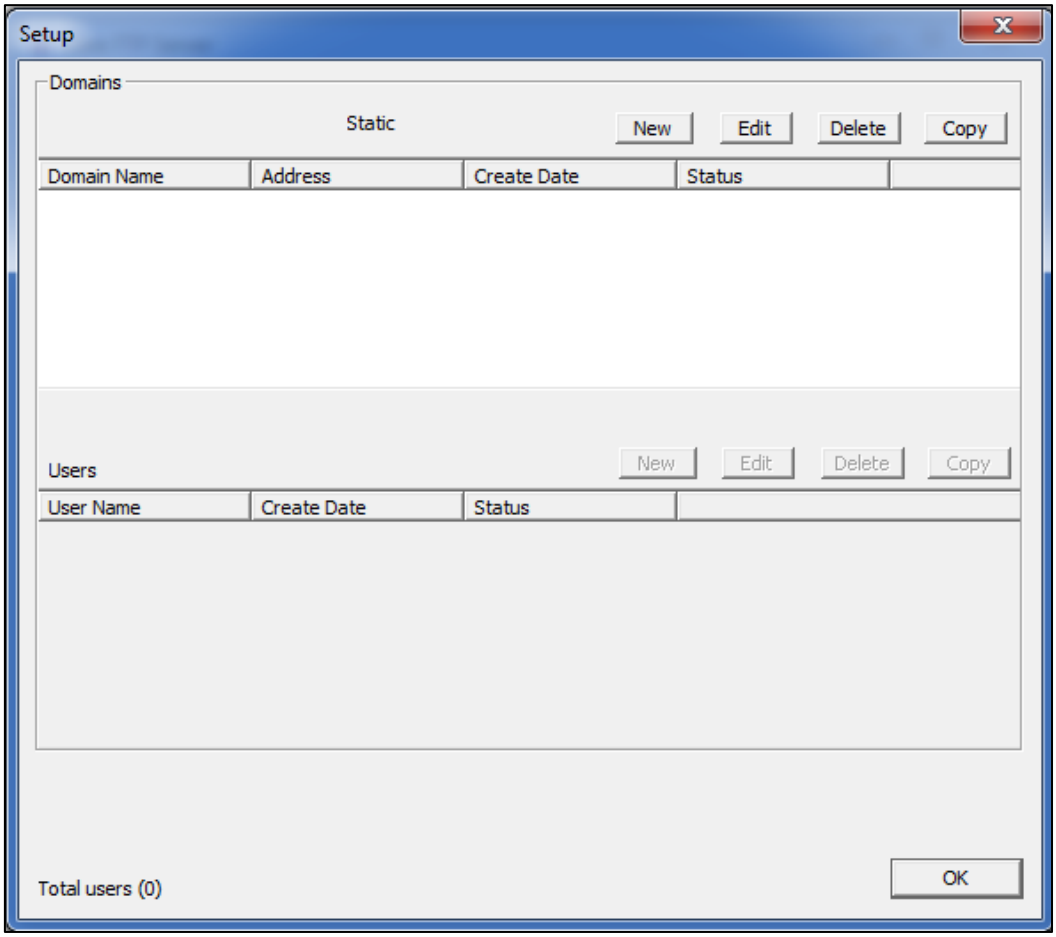


Figura 133. Ventana *setup* del servidor Core FTP.

Creación de un servidor SFTP

Dentro de *Setup* se pulsa el botón de *New* dentro del apartado *Domains* para incluir un nuevo servidor, se abre entonces una nueva interfaz en donde poder configurar todas las opciones del nuevo servidor.

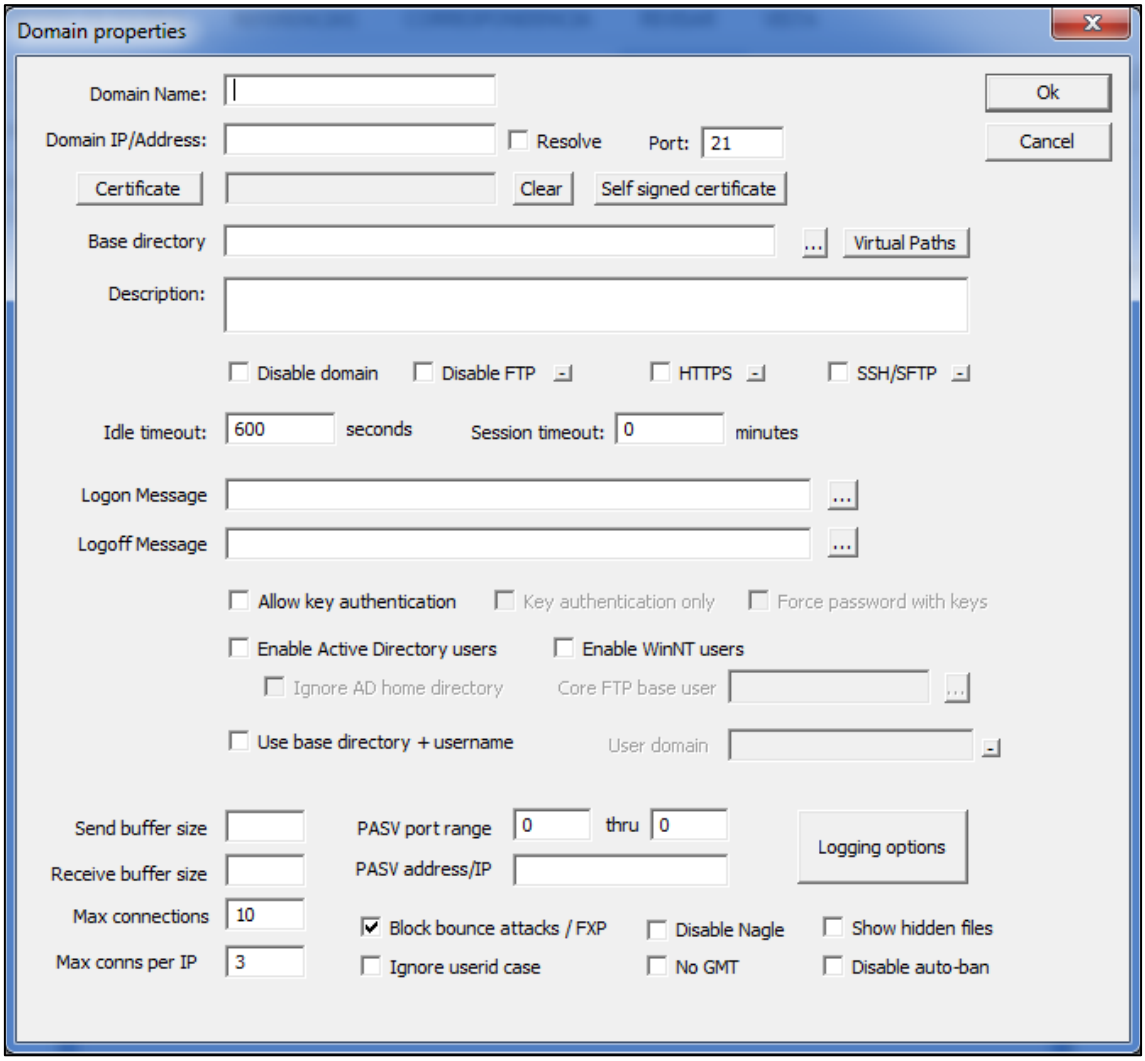


Figura 134. Ventana para la creación de un servidor.

A continuación se presentan cada una de las diferentes opciones disponibles, explicando brevemente para que sirven y cuál será el valor que tendrán que poner para el caso concreto que se está viendo.

- *Domain Name*: nombre del dominio para el nuevo servidor que se va a crear, puede ser un dominio o solo un nombre para el servidor.

- *Domain IP/Address*: dirección IP que se ha configurado en el equipo dentro de la red privada DVB-S. Si la red a la que está conectado el equipo es pública, es decir, la proporciona un ISP como Telefónica u otro, se podrá saber la dirección IP entrando en la página web http://www.cual-es-mi-ip.net/, aunque hay muchas más.

- *Port*: puerto que se usa para conectar con la aplicación. El puerto por defecto para las conexiones FTP es el 21 y en el caso de SFTP es el 22, en este caso se tendrá que poner el puerto 22.

  Importante: En el caso de que el equipo esté conectado a un router o cortafuegos, se tiene que abrir el puerto que ha configurado en el servidor SFTP, tanto en el router como en el cortafuegos.

- *Base directory*: carpeta raíz que se va a compartir con los clientes SFTP que se conectan al servidor. Cuando los clientes se conecten al equipo, verán el contenido de la carpeta seleccionada en este apartado.

- *Description*: descripción del servidor.

- *Idle timeout*: tiempo máximo de conexión con el servidor cuando el cliente no realiza ninguna acción en él.

- *Disable domain*: desactiva el dominio.

- *Disable FTP*: desactiva el modo FTP. Hay que desactivarlo en caso de que se configure un servidor SFTP u HTTPS.

- HTTPS: se configura como un servidor HTTPS. Hay que desactivarlo en caso de que se configure un servidor FTP u SFTP.

- SSH/SFTP: el servidor se crea como SFTP. Hay que desactivarlo en caso de que se configure un servidor FTP u HTTPS.

- *Session timeout*: tiempo máximo de conexión por sesión.

- *Loggon message*: mensaje que se muestra en el cliente cuando se conecta al SFTP.

- *Loggoff message*: mensaje que se muestra en el cliente cuando se desconecta del SFTP.

- *Allow key authentication*: opción que permite conectar con el servidor mediante unas claves de acceso DSA o RSA. Si se activa está opción el servidor tendrá que tener una clave pública y el cliente una clave privada.

- *Key authenticacion only*: autenticación del cliente con el servidor solo con claves DSA o RSA. Si se tuviera un usuario y contraseña, el servidor los descartaría. Solo se permite activar está opción si está activada la anterior.

- *Force password with keys*: Autenticación con claves DSA/RSA y contraseña del usuario que se conecta al servidor. Solo se permite activar está opción si está activada la anterior.

- *Enable active directory users*: habilita a los usuarios de un dominio *Active Directory* de Windows.

- *Ignore AD home directory*: ignora el directorio raíz del dominio de Active Directory.

- *Enable WinNT users*: habilita a los usuarios de un dominio WinNT.

- *Core FTP Base user*: permite elegir un usuario que se haya creado en el dominio del FTP/SFTP.

- *User Domain*: dominio de un *Active Directory* para asociarlo al usuario del servidor FTP/SFTP seleccionado en la opción anterior.

- *Use base directory + username*: usa como raíz del directorio en la conexión con el servidor FTP/SFTP una carpeta con el nombre del usuario dentro de la raíz del directorio que se ha definido en *Base directory*.

- *Send buffer size*: define el tamaño en bits de los paquetes que se envían.

- *Receive buffer size*: define el tamaño en bits de los paquetes que se reciben.

- *Max connections*: define el máximo de conexiones simultaneas que permite el servidor FTP/SFTP.

- *Max connections per IP*: define el máximo de conexiones simultaneas que permite el servidor FTP/SFTP desde una misma IP.

- *PASV port range*: rango de puertos que el servidor indica al cliente que puede usar el propio servidor para realizar la conexión pasiva.

- *PASV address/IP*: permite recibir todas las peticiones de la IP definida.

Las últimas opciones de la ventana de configuración son irrelevantes para la creación de un servidor SFTP, por lo que no se hace mención a ellas.

Creación de un usuario en un servidor SFTP

En el menú *Setup* se puede crear un usuario y configurarlo para que posteriormente el cliente SFTP pueda conectarse con él. Para la creación de usuarios es necesario que primero se haya creado el servidor, en caso contrario esta opción estará deshabilitada. Para crear un usuario se debe pinchar en el servidor y después pinchar en el botón *New* del apartador *Users*.

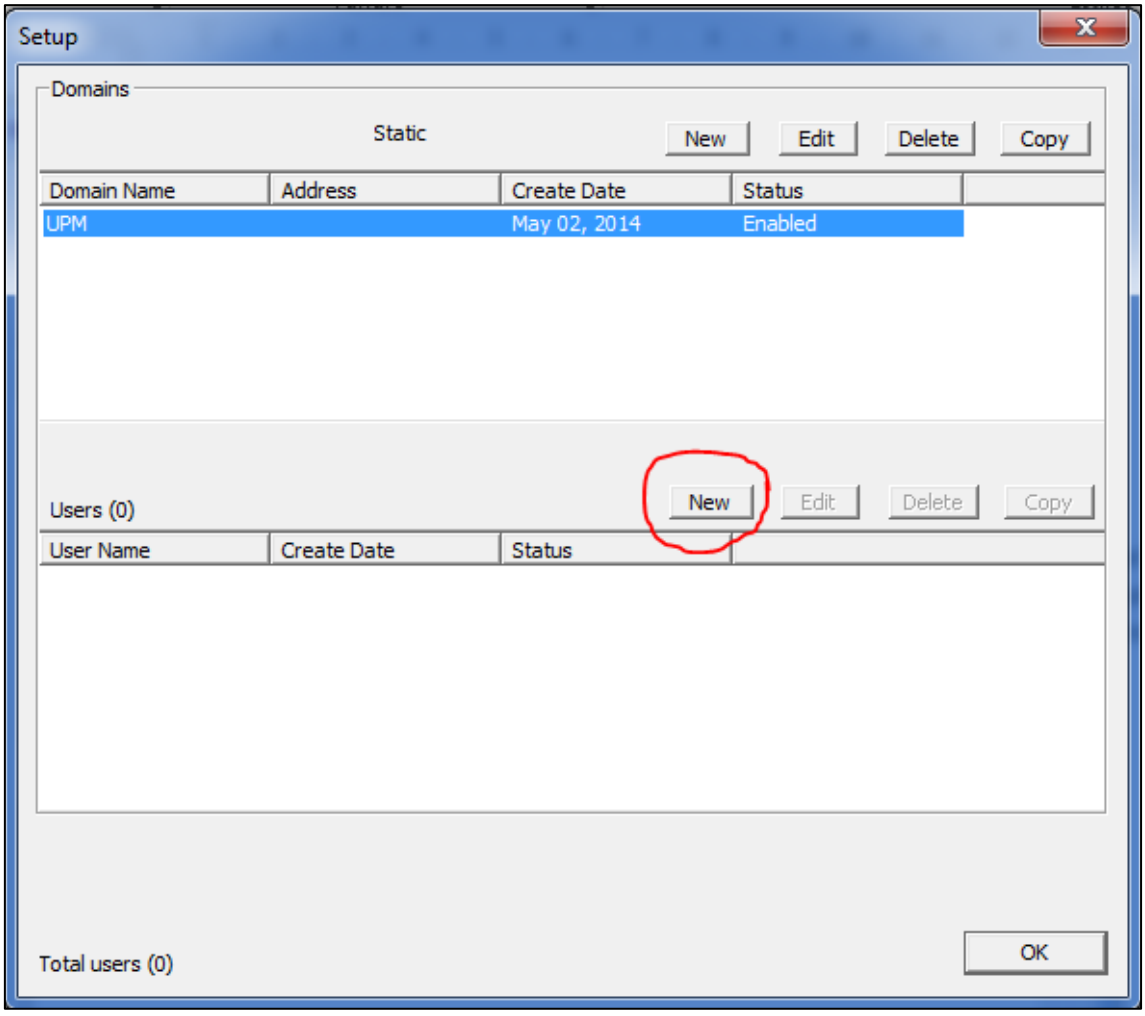


Figura 135. Ventana para *setup* para crear un usuario.

Se abrirá una ventana como se muestra en la siguiente imagen.

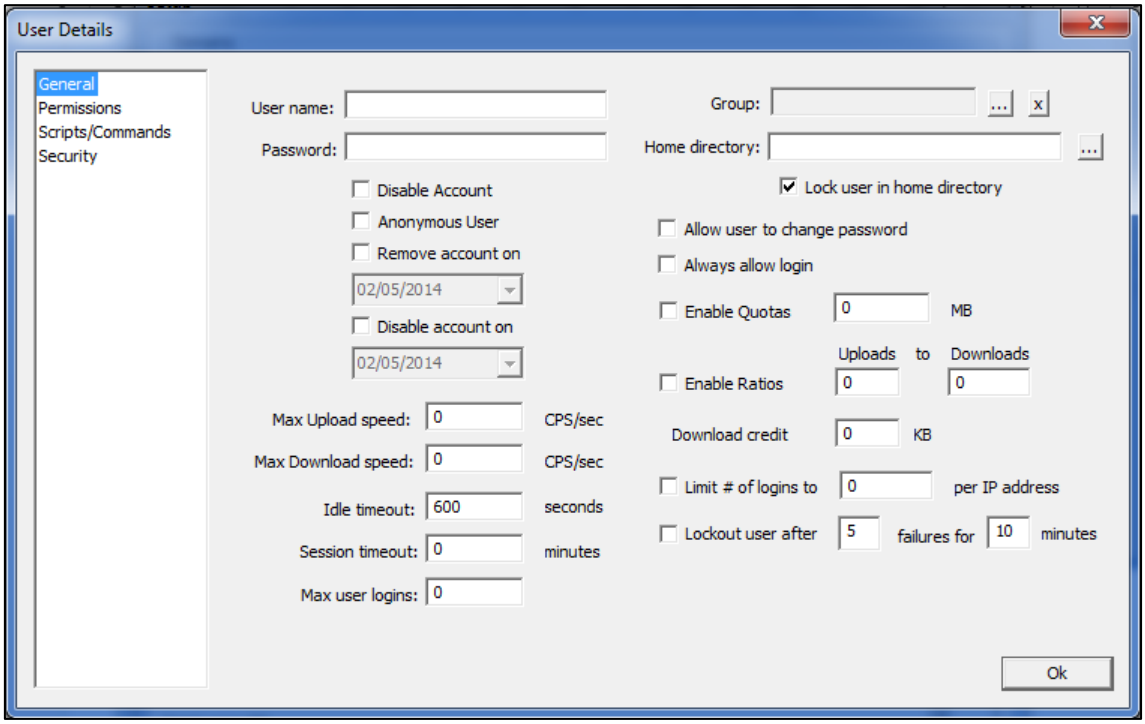


Figura 136. Ventana para la creación de un usuario en un servidor.

General

- *User Name*: se define el nombre de usuario.
- *Pasword*: se define una contraseña asociada al usuario.
- *Gruop*: si hay creado uno o varios grupo, se podrá asociar el usuario a un grupo.
- *Home Directory*: directorio raíz asociado al usuario.
- *Disable Account*: la cuenta del usuario queda deshabilitada.
- *Anonymous User*: al activar esta opción, al cliente no le hace falta usuario y contraseña.
- *Remove account on*: fija una fecha para eliminar el usuario.
- *Disable account on*: fija una fecha para deshabilitar el usuario.
- *Max upload speed*: máxima velocidad de subida de archivos que se le permite al usuario.
- *Max download speed*: máxima velocidad de descarga de archivos que se le permite al usuario.
- *Idle timeout*: tiempo máximo antes de desconectar al usuario del servidor si no se realiza ninguna acción.

- *Session timeout*: tiempo máximo de una sesión entre cliente y servidor.
- *Max user logins*: máximas conexiones que permite el servidor con el mismo usuario.
- *Allow user to change the password*: permite al usuario cambiar la contraseña.
- *Always allow login*: fuerza para que siempre permita la conexión.
- *Enable Quotas*: habita el espacio en MB que podrá usar el usuario.
- *Enable ratios*: habilita la opción de restringir la descarga de datos al usuario si no cumple el ratio.
- *Download credit*: fija un máximo de datos a descargar en la cuenta.
- *Limit # of logins to per IP Address*: limita el número de conexiones del usuario desde la misma IP.
- *Lockout user after failure for minutes*: bloquea un usuario durante un tiempo al loguearse de forma fallida un determinado número de veces.

Permissions

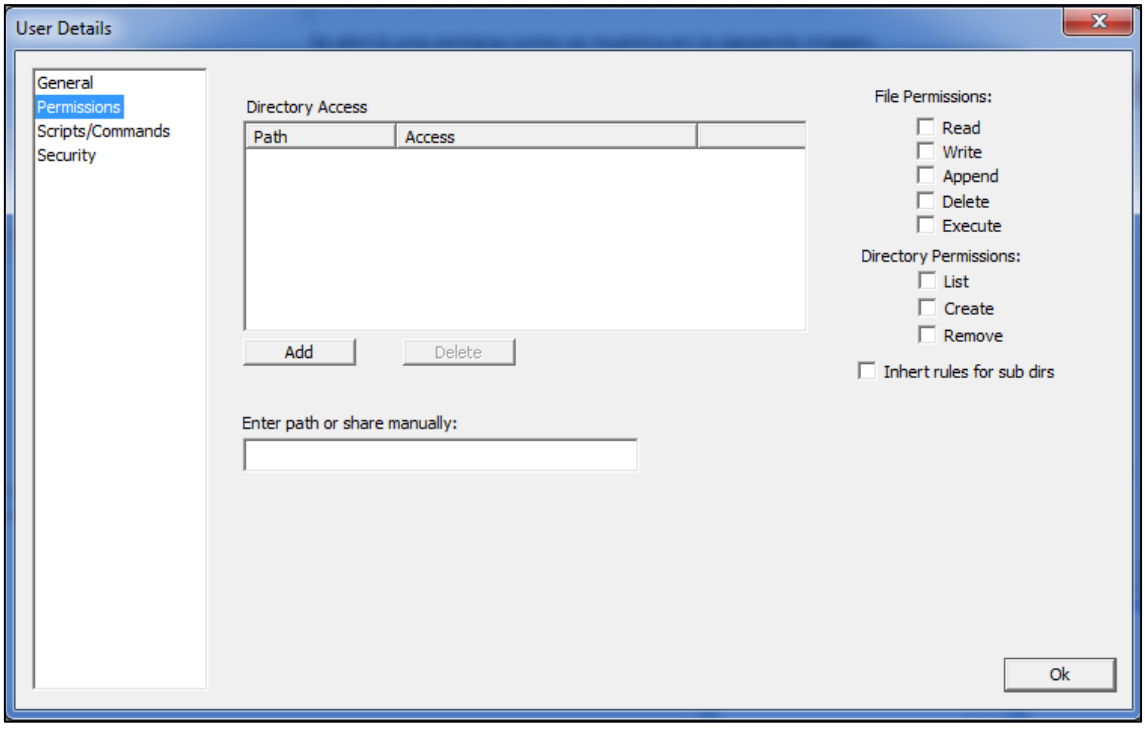


Figura 137. Ventana de permisos de un usuario.

Permite añadir un directorio de acceso, para cuando un cliente acceda al servidor SFTP con el usuario definido anteriormente tenga una serie de privilegios. Lo primero será

añadir un directorio pulsando el botón *Add*, cuando se haya incluido el directorio se podrán elegir los privilegios que tendrá el usuario en los ficheros y carpetas seleccionando las opciones de la derecha.

Privilegios en los ficheros:

- *Read*: permite leer/descargar el fichero.
- *Write*: permite modificar el fichero.
- *Append*: permite añadir un fichero.
- *Delete*: permite eliminar un fichero.
- *Execute*: permite ejecutar un fichero.

Privilegios en los directorios:

- *List*: permite que el cliente visualice los ficheros del directorio.
- *Create*: permite crear un elemento.
- *Remove*: permite eliminar una carpeta.

La opción *Inhert rules for subs dirs* también es importante, ya que permite aplicar estos permisos a los subdirectorios, es decir, a las carpetas que estén dentro del directorio principal seleccionado, para ello se debe activa dicha opción.

Scripts/Commands

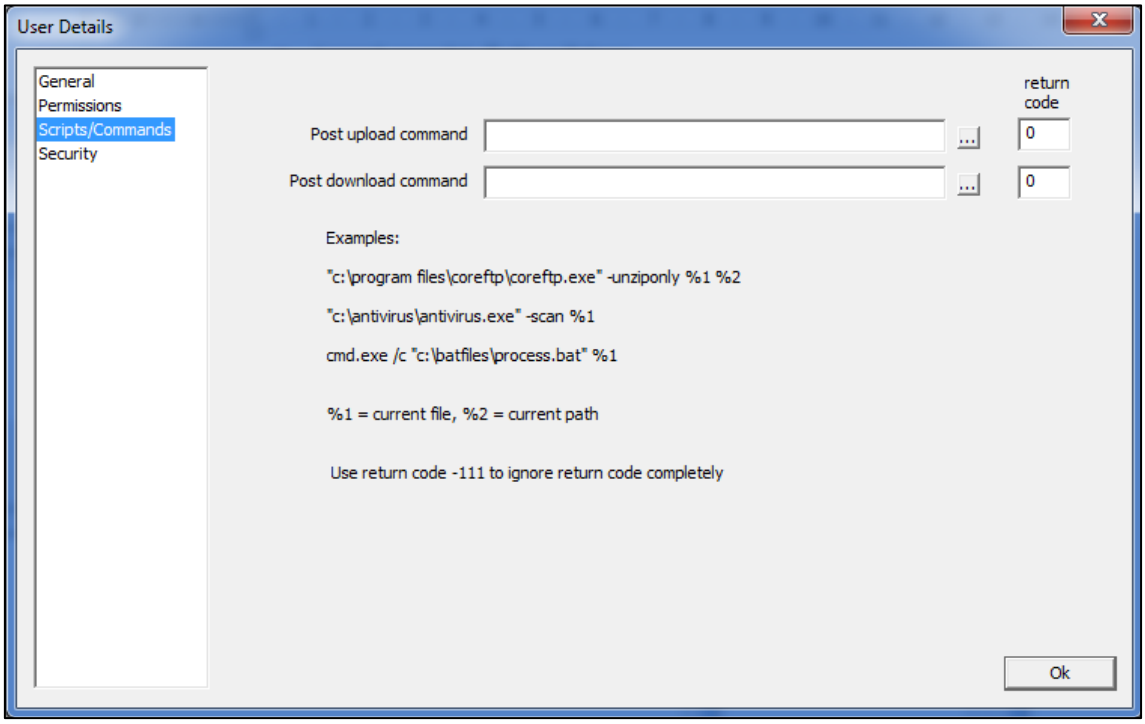


Figura 138. Ventana *Scrips/Commands* de un usuario.

En este apartado se pueden añadir comandos y scripts para realizar ciertas acciones con ficheros o carpetas, para ello se introducen los scripts en *Post upload command* o programar scripts y generar un ejecutable en la opción *Post download command*.

Security

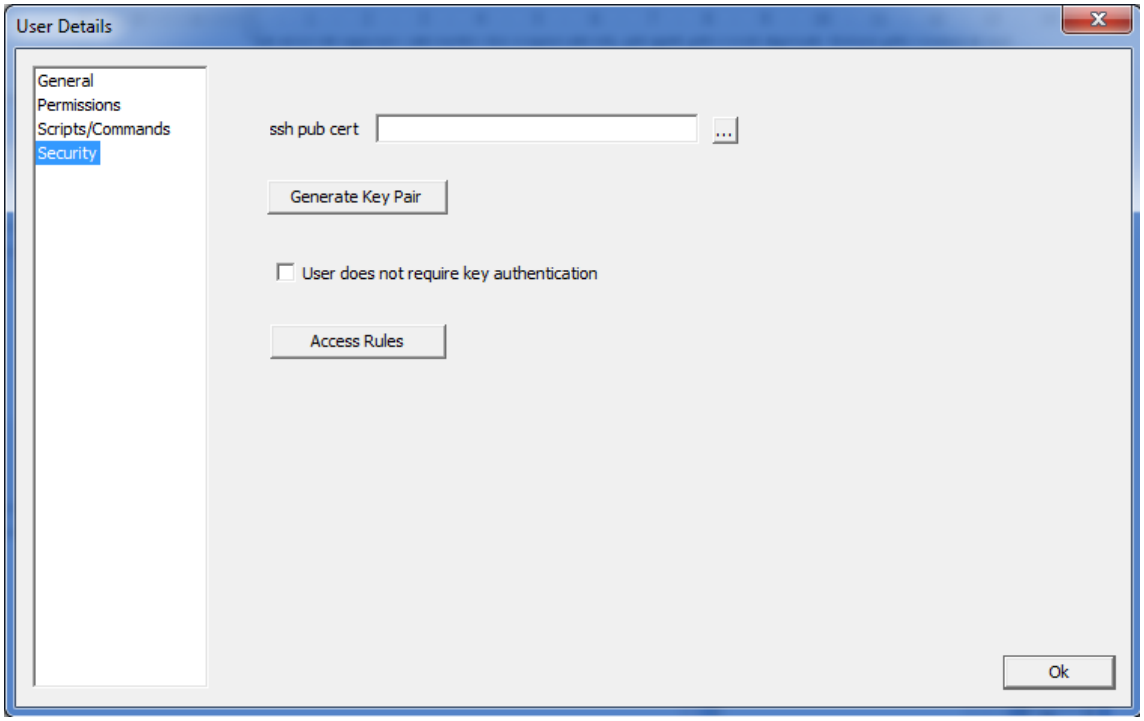


Figura 139. Ventana de seguridad de un usuario.

En esta última opción se puede configurar la seguridad de la conexión, para ello el propio servidor puede generar un par de claves DSA o RSA para realizar una conexión, y que el usuario y contraseña no sea el único requisito de identidad. Cuando se genere el par de claves por parte del servidor, la clave pública se deberá poner en el apartado *ssh pub cert* del servidor y la clave privada en el cliente.

A parte de esta característica, en la opción *Access Rules* también se puede añadir aún más seguridad. Dentro de esta opción se puede filtrar las direcciones IP que se pueden conectar con el usuario al servidor SFTP, prohibir direcciones IP, permitir o bloquear.

CÓDIGO APLICACIÓN CLIENTE SFTP

Clase Principal.cs

```
using System;
using System.IO;
using System.Collections;
using System.Net;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Configuration;
using Renci.SshNet;
using System.Globalization;

namespace SAICA
{
    public partial class Principal : Form
    {
        string directorio, dirEstacion, nmEstacion;
        string[] ficheros;
        string[,] extension = new string[,] { { ".mdb", "0" }, { ".xls", "2" }, {
".htm", "4" }, { ".pst", "5" }, { ".pdf", "6" }, { ".ppt", "7" }, { ".rar", "8"
}, { ".zip", "8" }, { ".txt", "9" }, { ".doc", "10" }, { ".gz", "8" }, { ".z",
"8" } };
        int itemSeleccionado;
        bool itemCarpeta, limpLog = false;
        List<KeyValuePair<string, string>> listFtp, descargarFtp;
        List<string> listLogTemp;
        ImageList listIconos = new ImageList();
        Timer timerDownload = new Timer();

        /*** Se inicializa el construtor ***/

        public Principal()
        {
            InitializeComponent();

            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;

            System.Drawing.Pen myPen;
            myPen = new System.Drawing.Pen(System.Drawing.Color.Red);
            System.Drawing.Graphics formGraphics = this.CreateGraphics();
            formGraphics.DrawLine(myPen, 0, 0, 200, 200);
            myPen.Dispose();
            formGraphics.Dispose();

            DirectoryInfo dirCarpeta = new DirectoryInfo(Application.StartupPath
+ @"\Resources\Iconos");
```

```
            foreach (FileInfo file in dirCarpeta.GetFiles())
            {
                listIconos.Images.Add(Image.FromFile(file.FullName));
            }

            timerDownload.Tick += new EventHandler(timer_Tick);

            /*** Muestra las carpetas (estaciones) en la ventana principal ***/
            listarEstaciones();
        }

        /*** Cierra el programa al seleccionar en el menú superior Archivo-
>cerrar ***/

        private void cerrarToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        /*** Abre una ventana de configuración de la aplicación ***/

        private void configuracionToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Configuracion config = new Configuracion();
            if (config.ShowDialog() == DialogResult.OK)
                listarEstaciones();  // Al cerrar la ventana de Configuración
vuelve a actualiar las carpetas (estaciones)
        }

        /*** Añade una opción en el menú superior con la información de la app
***/

        private void acercaDeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AcercaDe acercaDe = new AcercaDe();
            acercaDe.Show();
        }

        /*** Muestra en la ventana principal todas las carpetas (estaciones) ***/

        public void listarEstaciones()
        {
            List<string> listCarpetas = new List<string>();
            textFechaInicio.Enabled = false;
            textFechaFin.Enabled = false;
            comboFecha.Enabled = false;
            itemCarpeta = true;
            ImageList listIconos = new ImageList();
            DirectoryInfo dirCarpeta = new DirectoryInfo(Application.StartupPath
+ @"\Resources\Iconos");
            foreach (FileInfo file in dirCarpeta.GetFiles())
            {
                listIconos.Images.Add(Image.FromFile(file.FullName));
            }
            directorio = Configuracion.devolverDirectorioLocal();
            listViewDeskopt.Items.Clear();
            listViewDeskopt.SmallImageList = listIconos;
            if (directorio != null && directorio.Length != 0)
            {
                butDescargar.Enabled = true;
                checkTimer.Enabled = true;
```

```
                listCarpetas = Configuracion.devolverEstacionesGuardadas();
                foreach (string str in listCarpetas)
                {

                    if (!Directory.Exists(string.Format("{0}\\{1}", directorio,
str)))
                        Directory.CreateDirectory(string.Format("{0}\\{1}",
directorio, str));
                    string[] nombreCarpeta = { "Estación",
Path.GetFileName(string.Format("{0}\\{1}", directorio, str)),
File.GetLastWriteTime(string.Format("{0}\\{1}", directorio, str)).ToString() };
                    ListViewItem item = new ListViewItem(nombreCarpeta);
                    item.ImageIndex = 1;
                    item.Tag = str;

                    listViewDeskopt.Items.Add(item);
                }
            }
        }

        /*** Muestra en la ventana principal los ficheros de una carpeta
(estación) a la que se la ha hecho doble click ***/

        public void listarFicherosEstacion(string directorioEstacion)
        {
            itemCarpeta = false;
            textFechaInicio.Enabled = false;
            textFechaFin.Enabled = false;
            comboFecha.Enabled = false;

            listViewDeskopt.Items.Clear();
            listViewDeskopt.SmallImageList = listIconos;
            ficheros = Directory.GetFiles(directorioEstacion);

            string[] primero = { "", string.Format("{0}  {1}", '\u2191', "Subir
de directorio"), "" };
            ListViewItem item = new ListViewItem(primero);
            listViewDeskopt.Items.Add(item);

            foreach (string file in ficheros)
            {
                int itemImagen = 3;
                string extCadena;
                string[] nombreFichero = new string[3];
                if (string.Equals(Path.GetExtension(file), ""))
                    nombreFichero[0] = " ";
                else
                    nombreFichero[0] = char.ToUpper(Path.GetExtension(file)[1]) +
Path.GetExtension(file).Substring(2, Path.GetExtension(file).Length - 2);
                nombreFichero[1] = Path.GetFileNameWithoutExtension(file);
                nombreFichero[2] = File.GetLastWriteTime(file).ToString();
                item = new ListViewItem(nombreFichero);
                if (Path.GetExtension(file).Length > 4)
                    extCadena = Path.GetExtension(file).Substring(0,
Path.GetExtension(file).Length - 1);
                else
                    extCadena = Path.GetExtension(file);
                for (int i = 0; i < extension.GetLength(0); i++)
                {
                    if (extCadena.ToLower().Equals(extension[i, 0]))
                    {
                        itemImagen = Convert.ToInt32(extension[i, 1]);
```

```
                    }
                }
                item.ImageIndex = itemImagen;
                item.Tag = file;
                listViewDeskopt.Items.Add(item);
            }
            foreach (string f in ficheros)
            {
                f.Remove(0, f.Length);
            }
        }

        /*** Cuando se hace doble click en una carpeta (estación) o fichero ***/

        private void listViewDeskopt_DoubleClick(object sender, EventArgs e)
        {
            if (listViewDeskopt.Items[itemSeleccionado].SubItems[0].Text ==
"Estación")
            {
                dirEstacion = string.Format("{0}\\{1}", directorio,
listViewDeskopt.Items[itemSeleccionado].SubItems[1].Text);
                listarFicherosEstacion(string.Format("{0}\\{1}", directorio,
listViewDeskopt.Items[itemSeleccionado].SubItems[1].Text));
            }
            else if (listViewDeskopt.Items[itemSeleccionado].SubItems[0].Text ==
"")
                listarEstaciones();
            else if (listViewDeskopt.Items[itemSeleccionado].SubItems[0].Text ==
"Txt")
                abrirArchivoTextoLista(string.Format("{0}\\{1}", directorio,
nmEstacion), string.Format("{0}.{1}",
listViewDeskopt.Items[itemSeleccionado].SubItems[1].Text,
listViewDeskopt.Items[itemSeleccionado].SubItems[0].Text.ToLower()));
            else if (listViewDeskopt.Items[itemSeleccionado].SubItems[0].Text !=
"Txt")
                MessageBox.Show("Archivo seleccionado no soportado por la
aplicación.");
        }

        /*** Abre un fichero de texto que contenga una trama SAICA ***/

        private void abrirArchivoTextoLista(string dEstacion, string
nombrefichero)
        {
            bool hayDatos = false;
            string datos;
            datos = nmEstacion;// +" " + nombrefichero;
            int i = 1;
            StreamReader objReader = new StreamReader(string.Format("{0}\\{1}",
dEstacion, nombrefichero));
            string sLine = "";
            ArrayList arrText = new ArrayList();
            while (sLine != null)
            {
                sLine = objReader.ReadLine();
                if (sLine != null)
                {
                    datos = string.Format("{0};{1}", datos, sLine);
                    hayDatos = true;
                }
                i++;
            }
```

```
            objReader.Close();
            if (!hayDatos)
                MessageBox.Show("No hay datos dentro del documento.");
            else
            {
                escribirLog(string.Format("Abriendo el archivo {0}\\{1}",
dEstacion, nombrefichero));
                Datos_Estacion datosEstacion = new Datos_Estacion(datos);
                if (datosEstacion.ShowDialog() == DialogResult.OK)
                    datosEstacion.Show();
            }
        }

        /*** Al pulsar el botón "Descargar ficheros" de la ventana principal
empieza la conexión con las estaciones ***/

        private void butDescargar_Click(object sender, EventArgs e)
        {
            int downFicheros = 0;  // contador de los ficheros descargados
            int i = 0;
            string[,] listEstaciones = new
string[Configuracion.devolverNumeroEstacion(), 8];
            listEstaciones = Configuracion.devolverDatosEstacionesGuardadas();
            descargarFtp = new List<KeyValuePair<string, string>>();
            descargarFtp.Clear();
            if (listEstaciones != null)  // Comprobación que hay estaciones en
Configuración
            {
                while (i < listEstaciones.GetLength(0))  // Bucle que entra en
cada estación
                {
                    if (!Directory.Exists(string.Format("{0}\\{1}", directorio,
listEstaciones[i, 0])))  // Comprobación que el directorio de la estación está
creado
                    {
                        Directory.CreateDirectory(string.Format("{0}\\{1}",
directorio, listEstaciones[i, 0]));  // Crea el directorio de la estación en
local
                    }
                    string[] listTmpEstacion = new string[8];
                    for (int j = 0; j < listEstaciones.GetLength(1); j++)
                        listTmpEstacion[j] = listEstaciones[i, j];
                    conectarEstacionSFtp(listTmpEstacion);  // Conecta con una
estación
                    foreach (KeyValuePair<string, string> file in descargarFtp)
                        downFicheros++;
                    i++;
                }
                int downEstaciones = i; // Contador de estaciones con ficheros
descargados
                MessageBox.Show(string.Format("Se han descargado {0} ficheros de
{1} estaciones.", downFicheros, downEstaciones));
                escribirLog(string.Format("Se han descargado {0} ficheros de {1}
estaciones.", downFicheros, downEstaciones));
            }
            listarEstaciones();  // Al terminar la descarga vuelve a actualiar la
ventana principal con las estaciones
            listViewLog.EnsureVisible(listViewLog.Items.Count - 1);
        }

        /*** Conecta con una estación SFTP y descarga los ficheros que se
necesitan ***/
```

```
        private void conectarEstacionSFtp(string[] datEstacion)
        {
            escribirLog(string.Format("Se está conectando con la estación {0}",
datEstacion[0]));

            listFtp = new List<KeyValuePair<string, string>>();
            var localPath = directorio + "\\" + datEstacion[0] + "\\";
            var methods = new List<AuthenticationMethod>();
            var client = new SftpClient(datEstacion[1],
Convert.ToInt32(datEstacion[2]), datEstacion[3], datEstacion[4]);
            if (datEstacion[5].Equals("Si"))
            {
                try
                {
                    var keyFile = new PrivateKeyFile(datEstacion[6],
datEstacion[7]);
                    var keyFiles = new[] { keyFile };
                    methods.Add(new
PrivateKeyAuthenticationMethod(datEstacion[3], keyFiles));
                    var con = new ConnectionInfo(datEstacion[1],
Convert.ToInt32(datEstacion[2]), datEstacion[3], methods.ToArray());
                    client = new SftpClient(con);
                }
                catch
                {
                    escribirLog("La clave privada RSA no es compatible o no se
encuentra en la ubicación especificada.");
                    return;
                }
            }

            using (client)
            {
                try
                {
                    client.Connect();
                    escribirLog(string.Format("Conexión establecida con la
estación {0}", datEstacion[0]));

                    var files = client.ListDirectory("/");
                    foreach (var file in files)
                        listFtp.Add(new KeyValuePair<string, string>(file.Name,
file.FullName));
                    comprobarArchivos(datEstacion[0]);
                    foreach (KeyValuePair<string, string> file in descargarFtp)
                    {
                        escribirLog(string.Format("Descargando el {0} de la
estación {1}.", file.Key, datEstacion[0]));
                        using (var fs = new FileStream(localPath + file.Key,
FileMode.Create))
                        {
                            client.DownloadFile(file.Value, fs);
                            fs.Close();
                        }
                    }
                    escribirLog(string.Format("Se han descargado {0} ficheros de
la estación {1}.", descargarFtp.Count, datEstacion[0]));
                }
                catch (Exception e)
                {
                    MessageBox.Show(e.ToString());
```

```
                    switch (e.ToString().Substring(e.ToString().IndexOf(":") + 2,
(e.ToString().IndexOf(".", e.ToString().IndexOf(":")) -
e.ToString().IndexOf(":")) - 1))
                    {
                        case "No suitable authentication method found to complete
authentication.":
                            escribirLog(string.Format("No se ha podido conectar
con la estación {0}, error: No se ha podido concectar con el servidor por
problemas de autentificación.", datEstacion[0]));
                            break;
                        default:
                            escribirLog(string.Format("No se ha podido conectar
con la estación {0}, error: {1}.", datEstacion[0],
e.ToString().Substring(e.ToString().IndexOf(":") + 2, (e.ToString().IndexOf(".",
e.ToString().IndexOf(":")) - e.ToString().IndexOf(":")) - 1)));
                            break;
                    }
                }
                client.Disconnect();
            }
        }

        /*** Comprueba que los archivos que hay en el servidor no han sido
transfereido previamente ***/

        private void comprobarArchivos(string estacion)
        {
            List<string> listFileTemporal = new List<string>();
            descargarFtp = new List<KeyValuePair<string, string>>();
            ficheros = Directory.GetFiles(string.Format("{0}\\{1}", directorio,
estacion));
            foreach (string file in ficheros)
                listFileTemporal.Add(Path.GetFileName(file));
            foreach (KeyValuePair<string, string> file in listFtp)
            {
                if (comprobarNombre(file.Key) &&
!listFileTemporal.Contains(file.Key))
                    descargarFtp.Add(new KeyValuePair<string, string>(file.Key,
file.Value));
            }
        }

        /*** Se comprueba que el nombre del fichero tenga la estructura "dd-MM-
yyyy hh-mm-ss" ***/

        private static bool comprobarNombre(string archivo)
        {
            DateTime dateTime;
            string format = "dd-MM-yyyy hh-mm-ss";
            if (char.Equals('E', archivo[0]))
            {
                if (char.IsNumber(archivo, 1) && char.IsNumber(archivo, 2) &&
char.IsNumber(archivo, 3))
                {
                    if (DateTime.TryParseExact(archivo.Substring(5, 19), format,
CultureInfo.InvariantCulture, DateTimeStyles.None, out dateTime))
                        return true;
                }
            }
            return false;
        }
```

```
        private void checkTimer_CheckedChanged(object sender, EventArgs e)
        {
            if (checkTimer.Checked)
            {
                int tiempoAuto = Configuracion.devolverTiempoDescarga();
                MessageBox.Show(string.Format("La descarga automática se ha
activado. Se descargarán los ficheros de las estaciones cada {0} minutos.",
tiempoAuto));
                timerDownload.Interval = (1000) * (60) * (tiempoAuto);
                timerDownload.Enabled = true;
                timerDownload.Start();
            }
            else
            {
                MessageBox.Show("Se ha detenido la descarga automática.");
                timerDownload.Stop();
                timerDownload.Enabled = false;
            }
        }

        void timer_Tick(object sender, EventArgs e)
        {
            butDescargar_Click(sender, e);
        }

        /*** Metodo que se ejecuta cuando se selecciona una carpeta o fichero en
al ventana principal ***/

        private void listViewDeskopt_ItemSelectionChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
        {
            if (itemCarpeta)
            {
                textFechaInicio.Enabled = true;
                nmEstacion = listViewDeskopt.Items[e.ItemIndex].SubItems[1].Text;
            }
            itemSeleccionado = e.ItemIndex;
        }

        // Busca un fichero por fecha //

        private void textFechaInicio_Click(object sender, EventArgs e)
        {
            monthCalendarInicio.Visible = true;
            monthCalendarInicio.Enabled = true;
            textFechaFin.Enabled = false;
            textFechaFin.Text = "dd/mm/aaaa";
        }

        private void monthCalendarInicio_VisibleChanged(object sender, EventArgs
e)
        {
            if (monthCalendarInicio.Visible == true)
                monthCalendarInicio.BringToFront();
        }

        /*** Activa la seleccion para la fecha fin ***/

        private void monthCalendarInicio_KeyDown(object sender, KeyEventArgs e)
        {
```

```
            if (e.KeyCode == Keys.Enter & monthCalendarInicio.Visible == true)
            {
                textFechaInicio.Text =
monthCalendarInicio.SelectionRange.Start.ToShortDateString();
                monthCalendarInicio.Visible = false;
                textFechaFin.Enabled = true;
                picDate1.BackgroundImage =
Image.FromFile("Resources/icoCalendarioSeleccionado.ico");
            }
            else if (e.KeyCode == Keys.Escape & monthCalendarInicio.Visible ==
true)
                monthCalendarInicio.Visible = false;
        }

        /*** Cambia el color del texto fecha inicio de la busqueda a negro ***/

        private void textFechaInicio_TextChanged(object sender, EventArgs e)
        {
            textFechaInicio.ForeColor = System.Drawing.Color.Black;
        }

        /*** Activa el calendario para seleccion una fecha fin ***/

        private void textFechaFin_Click(object sender, EventArgs e)
        {
            monthCalendarFin.Visible = true;
        }

        /*** Trae al frente el calendario ***/

        private void monthCalendarFin_VisibleChanged(object sender, EventArgs e)
        {
            if (monthCalendarFin.Visible == true)
                monthCalendarFin.BringToFront();
        }

        /*** Comprueba que la fecha Fin de la busqueda es posterior a la fecha
inicio ***/

        private void monthCalendarFin_DateSelected(object sender,
DateRangeEventArgs e)
        {
            string fechaFinTemp =
monthCalendarFin.SelectionRange.Start.ToShortDateString();
            if (Convert.ToDateTime(fechaFinTemp) <
Convert.ToDateTime(textFechaInicio.Text))
            {
                MessageBox.Show("Selecciona una fecha mayor que la del incio de
la busqueda.");
            }
        }

        /*** Activa el despegable con los resultados de la busqueda ***/

        private void monthCalendarFin_KeyDown(object sender, KeyEventArgs e)
        {
            comboFecha.Items.Clear();
            comboFecha.Text = "";
            if (e.KeyCode == Keys.Escape & monthCalendarFin.Visible == true)
            {
                monthCalendarFin.Visible = false;
            }
```

```
            else if (e.KeyCode == Keys.Enter & monthCalendarFin.Visible == true)
            {
                textFechaFin.Text =
monthCalendarFin.SelectionRange.Start.ToShortDateString();
                monthCalendarFin.Visible = false;
                picDate2.BackgroundImage =
Image.FromFile("Resources/icoCalendarioSeleccionado.ico");
                comboFecha.Enabled = true;
                añadirFicherosBusqueda();
            }
        }

        /*** Cambia el color del texto fecha fin de la busqueda a negro ***/

        private void textFechaFin_TextChanged(object sender, EventArgs e)
        {
            textFechaFin.ForeColor = System.Drawing.Color.Black;
        }

        /*** Aañade los ficheros al despegable que cumplan las fechas
seleccionadas ***/

        private void añadirFicherosBusqueda()
        {

            List<KeyValuePair<string, DateTime>> listaComboTemporal = new
List<KeyValuePair<string, DateTime>>();
            ficheros = Directory.GetFiles(string.Format("{0}\\{1}", directorio,
listViewDeskopt.Items[itemSeleccionado].SubItems[1].Text));
            foreach (string file in ficheros)
            {
                if (Path.GetExtension(file).Equals(".txt"))
                    listaComboTemporal.Add(new KeyValuePair<string,
DateTime>(Path.GetFileNameWithoutExtension(file), File.GetLastWriteTime(file)));
            }
            if (listaComboTemporal.Count != 0)
            {
                foreach (KeyValuePair<string, DateTime> str in
listaComboTemporal)
                {
                    try
                    {
                        int separador = str.Key.IndexOf(" ", 5);
                        if
((DateTime.Compare(Convert.ToDateTime(str.Key.Substring(5, separador -
5).Replace("-", "/")), Convert.ToDateTime(textFechaInicio.Text)) >= 0) &&
(DateTime.Compare(Convert.ToDateTime(str.Key.Substring(5, separador -
5).Replace("-", "/")), Convert.ToDateTime(textFechaFin.Text)) <= 0))
                            comboFecha.Items.Add(str.Key);
                    }
                    catch { }
                }
            }
            if (comboFecha.Items.Count == 0)
            {
                comboFecha.Enabled = false;
                MessageBox.Show("No hay datos con los filtros selecionados.");
            }
        }

        /*** Activa el botón "Ver datos de la estación" en la pantalla principal
al seleccionar una fecha fin en
```

```
         * la busqueda ***/

        private void comboFecha_SelectedValueChanged(object sender, EventArgs e)
        {
            butDatosEstacion.Enabled = true;
        }

        /*** Al pulsar el botón "ver datos de la estación se llama al metodo
abrirArchivoTextoLista ***/

        private void butDatosEstacion_Click(object sender, EventArgs e)
        {
            abrirArchivoTextoLista(string.Format("{0}\\{1}", directorio,
nmEstacion), string.Format("{0}.txt", comboFecha.SelectedItem.ToString()));
        }

        /*** Se guarda en una lista los datos de entrada ***/

        private void escribirLog(string cadena)
        {
            string[] cadenaLog = { DateTime.Now.ToString(), cadena };
            ListViewItem item = new ListViewItem(cadenaLog);
            listViewLog.Items.Add(item);
        }

        /*** Se vacia el Log ***/

        private void butLimLog_Click(object sender, EventArgs e)
        {
            if (!limpLog)
                listLogTemp = new List<string>();
            if (listViewLog.Items.Count != 0)
            {
                foreach (ListViewItem item in listViewLog.Items)
                    listLogTemp.Add(string.Format("{0}   {1}",
item.SubItems[0].Text.ToString(), item.SubItems[1].Text.ToString()));
                listViewLog.Items.Clear();
                limpLog = true;
            }
        }

        /*** Guarda en un archivo de texto todo el log que ha generado la sesión
***/

        void guardarLog()
        {
            FileStream fsOut = File.Open(string.Format("{0}\\SFTP SAICA fecha {1}
hora {2}.txt", directorio, DateTime.Now.ToString("dd-MM-yyyy"),
DateTime.Now.ToString("HH-mm-ss")), FileMode.OpenOrCreate, FileAccess.Write);
            StreamWriter sw = new StreamWriter(fsOut);
            sw.WriteLine("");
            sw.WriteLine("*************************************************");
            sw.WriteLine("***                                           ***");
            sw.WriteLine(string.Format("*** LOG SFTP SAICA a fecha {0} ***",
DateTime.Now.ToString()));
            sw.WriteLine("***                                           ***");
            sw.WriteLine("*************************************************");
            sw.WriteLine("");

            if (limpLog)
            {
                foreach (string cadena in listLogTemp)
```

```
                    sw.WriteLine(cadena + "");
            }
            foreach (ListViewItem item in listViewLog.Items)
                sw.WriteLine(string.Format("{0}   {1}", item.SubItems[0].Text,
item.SubItems[1].Text));

            sw.Close();
            fsOut.Close();
            if (listViewLog.Items.Count != 0)
                listViewLog.EnsureVisible(listViewLog.Items.Count - 1);
        }

        /*** Se guarda el LOG al pulsar la X de la ventana ***/

        private void Principal_FormClosing(object sender, FormClosingEventArgs e)
        {
            DialogResult dialogResult = MessageBox.Show("¿Está seguro que quiere
salir de la aplicación?", "Salir de la aplicación", MessageBoxButtons.YesNo);
            if (dialogResult == DialogResult.Yes)
                guardarLog(); // Llamada al metodo de guardarLog
            else if (dialogResult == DialogResult.No)
            {
                e.Cancel = true;
            }
        }
    }
}
```

Clase Configuracion.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Configuration;
using System.IO;

namespace SAICA
{
    public partial class Configuracion : Form
    {
        public string pathPrivateKey;
        static string pathEmpty = "No se ha seleccionado ningún directorio";
        public Configuration config;
        public string[] nuevaEstacion;
        public int itemSeleccionado;

        public Configuracion()
        {
            InitializeComponent();

            config =
ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
```

```
            try
            {
                textLocalPath.Text =
config.AppSettings.Settings["PathLocalDirectory"].Value;
            }
            catch
            {
                textLocalPath.Text = Application.StartupPath + "\\Descargas";
                if (!Directory.Exists(string.Format("{0}",
textLocalPath.Text.ToString())))
                    Directory.CreateDirectory(string.Format("{0}",
textLocalPath.Text.ToString()));
            }
            try
            {
                if (config.AppSettings.Settings["Conexión
pasiva"].Value.Equals("True"))
                    checkPassiveMode.Checked = true;
                else
                    checkPassiveMode.Checked = false;
            }
            catch
            {
                checkPassiveMode.Checked = false;
            }
            try
            {
                numericAutoDown.Value =
Convert.ToInt32(config.AppSettings.Settings["Tiempo descarga automatica"].Value);
            }
            catch
            {
                numericAutoDown.Value = 10;
            }
            listarEstaciones();
        }

        /*** Abre una ventana emergente para seleccionar una carpeta local dónde
guardar los ficheros ***/

        private void butDirectorio_Click(object sender, EventArgs e)
        {
            foreach (ListViewItem item in listEstaciones.Items)
                item.Selected = false;
            butEditEstacion.Enabled = false;
            butDeleteEstacion.Enabled = false;
            FolderBrowserDialog folderPicker = new FolderBrowserDialog();
            if (folderPicker.ShowDialog() == DialogResult.OK)
                textLocalPath.Text = folderPicker.SelectedPath;
        }

        /*** Guarda la configuración ***/

        private void guardar_config_Click(object sender, EventArgs e)
        {
            config.AppSettings.Settings.Remove("PathLocalDirectory");
            if (!textLocalPath.Text.Equals(pathEmpty))
                config.AppSettings.Settings.Add("PathLocalDirectory",
textLocalPath.Text);
            config.AppSettings.Settings.Remove("Conexión pasiva");
            config.AppSettings.Settings.Add("Conexión pasiva",
checkPassiveMode.Checked.ToString());
```

```
            config.AppSettings.Settings.Remove("Tiempo descarga automatica");
            config.AppSettings.Settings.Add("Tiempo descarga automatica",
numericAutoDown.Value.ToString());
            guardarEstaciones();
            config.Save(ConfigurationSaveMode.Modified);
            ConfigurationManager.RefreshSection("appSettings");
            DialogResult = DialogResult.OK;
            this.Close();
        }

        /*** Cierra la ventana de configuración sin guardar ***/

        private void cancelar_config_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        public static string devolverDirectorioLocal()
        {
            if
(string.IsNullOrEmpty(ConfigurationManager.AppSettings["PathLocalDirectory"]))
                return null;
            else
                return ConfigurationManager.AppSettings["PathLocalDirectory"];
        }

        public static string devolverPrivateKey(string nomEstacion)
        {
            int i = 0;
            string pathKeyEstacion = null;
            while
(!string.IsNullOrEmpty(ConfigurationManager.AppSettings["nombreEstacion " +
i.ToString()]))
            {
                if (ConfigurationManager.AppSettings["nombreEstacion " +
i.ToString()].Equals(nomEstacion))
                    pathKeyEstacion =
ConfigurationManager.AppSettings["PathPrivateKey " + i.ToString()];
                i++;
            }
            return pathKeyEstacion;
        }

        /*** Muestra las estaciones que hay guardadas en la aplicación ***/

        public void listarEstaciones()
        {
            string[] estacion;
            listEstaciones.Items.Clear();
            int i = 0;
            while (config.AppSettings.Settings["nombreEstacion " + i.ToString()]
!= null)
            {
                estacion = new string[8] {
config.AppSettings.Settings["nombreEstacion " +
                    i.ToString()].Value,
config.AppSettings.Settings["dirIpEstacion " + i.ToString()]
                    .Value, config.AppSettings.Settings["puertoEstacion " +
i.ToString()].Value, config
                    .AppSettings.Settings["usuarioEstacion " +
i.ToString()].Value, config.AppSettings
```

```
                    .Settings["passwordEstacion " + i.ToString()].Value,
config.AppSettings.Settings
                    ["privateKey " + i.ToString()].Value,
config.AppSettings.Settings["PathPrivateKey "
                    + i.ToString()].Value, config.AppSettings.Settings["Clave de
paso " + i.ToString()].Value };
                ListViewItem item = new ListViewItem(estacion);

                listEstaciones.Items.Add(item);
                i++;
            }
        }

        /*** Abre una ventana nueva para añadir una estación ***/

        private void butAddEstacion_Click(object sender, EventArgs e)
        {
            AddEstacion estacion = new AddEstacion(1);
            estacion.nombresEstacion = new List<String>();
            foreach (ListViewItem item in listEstaciones.Items)
            {
                estacion.nombresEstacion.Add(item.SubItems[0].Text);
            }
            if (estacion.ShowDialog() == DialogResult.OK)
            {
                nuevaEstacion = new string[8];
                estacion.datosEstacion.CopyTo(nuevaEstacion, 0);
                añadirEstacion(nuevaEstacion);
            }
            listEstaciones.Items[itemSeleccionado].Selected = false;
            butDeleteEstacion.Enabled = false;
            butEditEstacion.Enabled = false;
        }

        /*** Abre una vetana nueva para editar una estación seleccionada ***/

        private void butEditEstacion_Click(object sender, EventArgs e)
        {
            AddEstacion estacion = new AddEstacion(0);
            int i;
            estacion.datosEstacion = new string[8];
            for (i = 0; i < estacion.datosEstacion.Length; i++)
                estacion.datosEstacion[i] =
listEstaciones.Items[itemSeleccionado].SubItems[i].Text;
            estacion.escribirEstacion();
            i = 0;
            estacion.nombresEstacion = new List<String>();
            while (i < listEstaciones.Items.Count)
            {
                if (i != itemSeleccionado)
                {

estacion.nombresEstacion.Add(listEstaciones.Items[i].SubItems[0].Text);
                }
                i++;
            }

            if (estacion.ShowDialog() == DialogResult.OK)
            {
                nuevaEstacion = new string[8];
                estacion.datosEstacion.CopyTo(nuevaEstacion, 0);
                ListViewItem item = new ListViewItem(nuevaEstacion);
```

```
                listEstaciones.Items[itemSeleccionado] = item;
            }
            listEstaciones.Items[itemSeleccionado].Selected = false;
            butDeleteEstacion.Enabled = false;
            butEditEstacion.Enabled = false;
        }


        private void añadirEstacion(string[] newEstacion)
        {
            ListViewItem item = new ListViewItem(newEstacion);
            listEstaciones.Items.Add(item);
        }

        private void guardarEstaciones()
        {
            int i = 0;
            while (config.AppSettings.Settings["nombreEstacion " + i.ToString()]
!= null)
            {
                config.AppSettings.Settings.Remove("nombreEstacion " +
i.ToString());
                config.AppSettings.Settings.Remove("dirIpEstacion " +
i.ToString());
                config.AppSettings.Settings.Remove("puertoEstacion " +
i.ToString());
                config.AppSettings.Settings.Remove("usuarioEstacion " +
i.ToString());
                config.AppSettings.Settings.Remove("passwordEstacion " +
i.ToString());
                config.AppSettings.Settings.Remove("privateKey " + i.ToString());
                config.AppSettings.Settings.Remove("PathPrivateKey " +
i.ToString());
                config.AppSettings.Settings.Remove("Clave de paso " +
i.ToString());
                i++;
            }
            i = 0;
            while (i < listEstaciones.Items.Count)
            {
                config.AppSettings.Settings.Add("nombreEstacion " + i.ToString(),
listEstaciones.Items[i].SubItems[0].Text);
                config.AppSettings.Settings.Add("dirIpEstacion " + i.ToString(),
listEstaciones.Items[i].SubItems[1].Text);
                config.AppSettings.Settings.Add("puertoEstacion " + i.ToString(),
listEstaciones.Items[i].SubItems[2].Text);
                config.AppSettings.Settings.Add("usuarioEstacion " +
i.ToString(), listEstaciones.Items[i].SubItems[3].Text);
                config.AppSettings.Settings.Add("passwordEstacion " +
i.ToString(), listEstaciones.Items[i].SubItems[4].Text);
                config.AppSettings.Settings.Add("privateKey " + i.ToString(),
listEstaciones.Items[i].SubItems[5].Text);
                config.AppSettings.Settings.Add("PathPrivateKey " + i.ToString(),
listEstaciones.Items[i].SubItems[6].Text);
                config.AppSettings.Settings.Add("Clave de paso " + i.ToString(),
listEstaciones.Items[i].SubItems[7].Text);
                i++;
            }
        }

        private void listEstaciones_ItemSelectionChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
```

```
        {
            itemSeleccionado = e.ItemIndex;
            butDeleteEstacion.Enabled = true;
            butEditEstacion.Enabled = true;
        }

        public static int devolverNumeroEstacion()
        {
            int i = 0;
            while (ConfigurationManager.AppSettings["nombreEstacion " +
i.ToString()] != null)
            {
                i++;
            }

            return i;
        }

        public static List<string> devolverEstacionesGuardadas()
        {
            List<string> listaEstaciones = new List<string>();
            int i = 0;
            if (devolverNumeroEstacion() != 0)
            {
                while (ConfigurationManager.AppSettings["nombreEstacion " +
i.ToString()] != null)
                {

listaEstaciones.Add(ConfigurationManager.AppSettings["nombreEstacion " +
i.ToString()]);
                    i++;
                }
                return listaEstaciones;
            }
            else
                return listaEstaciones = new List<string>();
        }

        public static string[,] devolverDatosEstacionesGuardadas()
        {
            int i = 0;
            string[,] listEstaciones = new string[devolverNumeroEstacion(), 8];
            if (devolverNumeroEstacion() != 0)
            {
                while (ConfigurationManager.AppSettings["nombreEstacion " +
i.ToString()] != null)
                {
                    listEstaciones[i, 0] =
ConfigurationManager.AppSettings["nombreEstacion " + i.ToString()];
                    listEstaciones[i, 1] =
ConfigurationManager.AppSettings["dirIpEstacion " + i.ToString()];
                    listEstaciones[i, 2] =
ConfigurationManager.AppSettings["puertoEstacion " + i.ToString()];
                    listEstaciones[i, 3] =
ConfigurationManager.AppSettings["usuarioEstacion " + i.ToString()];
                    listEstaciones[i, 4] =
ConfigurationManager.AppSettings["passwordEstacion " + i.ToString()];
                    listEstaciones[i, 5] =
ConfigurationManager.AppSettings["privateKey " + i.ToString()];
                    listEstaciones[i, 6] =
ConfigurationManager.AppSettings["PathPrivateKey " + i.ToString()];
```

```
                    listEstaciones[i, 7] =
ConfigurationManager.AppSettings["Clave de paso " + i.ToString()];
                    i++;
                }
                return listEstaciones;
            }
            else
            {
                return listEstaciones = null;
            }
        }

        public static int devolverTiempoDescarga()
        {
            int tiempo;
            tiempo = Convert.ToInt32(ConfigurationManager.AppSettings["Tiempo
descarga automatica"]);
            return tiempo;
        }

        /*** Borra ua estación seleccionada ***/

        private void butDeleteEstacion_Click(object sender, EventArgs e)
        {
            listEstaciones.Items[itemSeleccionado].Remove();
            butDeleteEstacion.Enabled = false;
            butEditEstacion.Enabled = false;
        }

        /*** Activa o descativa el tipo de conexión pasiva ***/

        private void checkPassiveMode_Click(object sender, EventArgs e)
        {
            foreach (ListViewItem item in listEstaciones.Items)
                item.Selected = false;
            butEditEstacion.Enabled = false;
            butDeleteEstacion.Enabled = false;
        }

        private void textLocalPath_Click(object sender, EventArgs e)
        {
            foreach (ListViewItem item in listEstaciones.Items)
                item.Selected = false;
            butEditEstacion.Enabled = false;
            butDeleteEstacion.Enabled = false;
        }
    }
}
```

Clase AddEstacion.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Windows.Forms;

namespace SAICA
{
    public partial class AddEstacion : Form
    {
        public string[] datosEstacion;
        string privateKey;
        public bool statusNombre, statusDirIp, statusPuerto;
        string errorNombre, errorDirIp, errorPuerto;
        public List<string> nombresEstacion;

        public AddEstacion(int opcion)
        {
            InitializeComponent();
            if (opcion == 1)  // Comprueba si abre el formulario para añadir una
estación
            {
                textPuerto.Text = "22";
                checkAnonimo.Checked = true;
                butDeleteKey.Enabled = false;
                textPrivateKey.Text = "No se ha seleccionado ningún directorio";
                textFrasePaso.Enabled = false;
                statusNombre = false;
                errorNombre = "No ha introducido ningún nombre en la estación.";
                statusDirIp = false;   // cambiado junto con leave DirIp
                errorDirIp = "No ha introducido ninguna dirección IP";
                statusPuerto = true;
                privateKey = "No";
                this.Text = "Añadir nueva estación";
                this.Icon = new System.Drawing.Icon("Resources/icoAñadir.ico");
            }
            else  // Opción para editar una estación creada
            {
                this.Text = "Editar estación";
                this.Icon = new System.Drawing.Icon("Resources/icoEditar.ico");
            }
        }

        /*** Escribe los datos de la estación en los campos de la ventana ***/

        public void escribirEstacion()
        {
            textEstacion.Text = datosEstacion[0];
            textDirIp.Text = datosEstacion[1];
            textPuerto.Text = datosEstacion[2];
            if (datosEstacion[3].Equals("anonymous") &&
datosEstacion[4].Equals("anonymous"))
                checkAnonimo.Checked = true;
            else
            {
                textUsuario.Text = datosEstacion[3];
                textPassword.Text = datosEstacion[4];
            }
            privateKey = datosEstacion[5];
            textFrasePaso.Enabled = false;
            textFrasePaso.Text = "";
            if (privateKey.Equals("No"))
            {
                butPrivateKey.Enabled = true;
                butDeleteKey.Enabled = false;
                textPrivateKey.Text = "No se ha seleccionado ningún directorio";
```

```
            }
            else
            {
                butDeleteKey.Enabled = true;
                textPrivateKey.Text = datosEstacion[6];
                textFrasePaso.Enabled = true;
                if (!String.IsNullOrEmpty(datosEstacion[7]))
                    textFrasePaso.Text = datosEstacion[7];
            }
            statusNombre = true;
            statusDirIp = true;
            statusPuerto = true;
        }

        private void textEstacion_Leave(object sender, EventArgs e)
        {
            statusNombre = false;
            if (textEstacion.Text.Length == 0)
                errorNombre = "Nombre de la estación vacio, introduzca un
nombre.";
            else if (nombresEstacion.Contains(textEstacion.Text))
                errorNombre = "Nombre de la estación ya usado, elija otro.";
            else
                statusNombre = true;
        }

        private void textDirIp_Leave(object sender, EventArgs e)
        {
            statusDirIp = false;
            if (textDirIp.Text.Length == 0)
                errorDirIp = "No hay introducido ninguna dirección IP.";
            else if (!comprobarDirIp(textDirIp.Text))
                errorDirIp = "Formato de la dirección IP erroneo, formato
correcto XXX.XXX.XXX.XXX. Cada XXX debe estar comprendido entre 0 y 255.";
            else
                statusDirIp = true;
        }

        /*** Comprueba que la dirección IP tenga el formato XXX.XXX.XXX.XXX ***/

        private bool comprobarDirIp(string direccionIp)
        {
            int i = 0, inicio, fin, direccionTemp;
            foreach (char c in direccionIp)
                if (c == '.')
                    i++;
            if (i != 3)
                return false;
            else
                fin = -1;
            for (i = 0; i < 4; i++)
            {
                inicio = fin + 1;
                if (i != 3)
                    fin = direccionIp.IndexOf('.', inicio);
                else
                    fin = direccionIp.Length;
                try
                {
                    direccionTemp = Convert.ToInt32(direccionIp.Substring(inicio,
fin - inicio));
                    if (direccionTemp < 0 || direccionTemp > 255)
```

```
                    return false;
                }
                catch
                {
                    return false;
                }
            }
            return true;
        }

        /*** Comprueba que el puerto de la estación sea un número ***/

        private void textPuerto_Leave(object sender, EventArgs e)
        {
            int number;
            statusPuerto = false;
            if (textPuerto.Text.Length == 0)
                errorPuerto = "Introduzca un número de puerto.";
            else if (!Int32.TryParse(textPuerto.Text, out number))
                errorPuerto = "El puerto debe ser numérico.";
            else if (Convert.ToInt32(textPuerto.Text) > 65535 ||
Convert.ToInt32(textPuerto.Text) < 0)
                errorPuerto = "El puerto debe estar comprendido entre el 0 y el
65535.";
            else
                statusPuerto = true;
        }

        /*** se abre una ventana de seleccion de ficheros para seleccionar una
clave privada ***/

        private void butPrivateKey_Click(object sender, EventArgs e)
        {
            OpenFileDialog filePicker = new OpenFileDialog();
            filePicker.Multiselect = false;
            FolderBrowserDialog folderPicker = new FolderBrowserDialog();
            if (filePicker.ShowDialog() == DialogResult.OK)
            {
                textPrivateKey.Text = filePicker.FileName;
                butDeleteKey.Enabled = true;
                privateKey = "Si";
                textFrasePaso.Enabled = true;
            }
        }

        /*** borra la clave privada que habia seleccionada ***/

        private void butDeleteKey_Click(object sender, EventArgs e)
        {
            privateKey = "No";
            textPrivateKey.Text = "No se ha seleccionado ningún archivo";
            butDeleteKey.Enabled = false;
            textFrasePaso.Text = "";
            textFrasePaso.Enabled = false;
        }

        /*** Cambiar el nombre del usuario y la contraseña si se activa el
checkbox de "Usuario y contraseña anónimos" ***/

        private void checkAnonimo_CheckedChanged(object sender, EventArgs e)
        {
            if (checkAnonimo.Checked) // Si el checkbox se activa
```

```
            {
                textUsuario.Text = "anonymous"; // En usuario se escribe
"anonymous"
                textUsuario.Enabled = false;
                textPassword.Text = "anonymous"; // En contraseña se escribe
"anonymous"
                textPassword.Enabled = false;
            }
            else // Si se desactiva el checkbox
            {
                textUsuario.Text = ""; // El nombre del usuario aparece vacio
                textUsuario.Enabled = true;
                textPassword.Text = ""; // La contraseña aparece vacia
                textPassword.Enabled = true;
            }
        }

        private void textUsuario_VisibleChanged(object sender, EventArgs e)
        {
            if (!textUsuario.Visible)
                textUsuario.Text = "";
        }

        private void textPassword_VisibleChanged(object sender, EventArgs e)
        {
            if (!textPassword.Visible)
                textPassword.Text = "";
        }

        /*** Cierra la ventana guardando la estación ***/

        private void butAceptarEstacion_Click(object sender, EventArgs e)
        {
            if (!statusNombre) // Se comprueba que el nombre de la estación no
esté vacio
                MessageBox.Show(errorNombre);
            else if (!statusDirIp) // Se comprueba que la dirección IP no esté
vacia
                MessageBox.Show(errorDirIp);
            else if (!statusPuerto) // Se comprueba que el puerto no esté vacio
                MessageBox.Show(errorPuerto);
            else
            {
                datosEstacion = new string[8] { textEstacion.Text,
textDirIp.Text, textPuerto.Text, textUsuario.Text, textPassword.Text, privateKey,
textPrivateKey.Text, textFrasePaso.Text };
                DialogResult = DialogResult.OK;
                this.Close();
            }
        }

        /*** Cierra la ventana sin guardar la estación ***/

        private void butCancelarEstacion_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Clase Datos Estacion.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SAICA
{
    public partial class Datos_Estacion : Form
    {
        int cont, estacionOrigen, estacionDestino, longDatos, numItems, diaSaica,
horaSaica, hora, minuto, resto;
        string tramaSaica, classType;
        DateTime dt = new DateTime(1920, 1, 1);
        DateTime hm1 = new DateTime(1900, 1, 1, 0, 0, 0);
        DateTime hm2 = new DateTime(1900, 1, 1, 0, 0, 0);

        public Datos_Estacion(string datos)
        {
            InitializeComponent();
            cont = datos.IndexOf(";");
            this.Text = datos.Substring(0, cont);
            tramaSaica = datos.Substring(cont);
            try
            {
                leerTramaSaica();
                convertirDiaHoraSaica();
                labelEstacion.Text = "Muestras tomadas en la estación de " + Text
+ " el día " + String.Format("{0:dd/MM/yyyy}", dt) + " entre las " +
hm1.ToString("HH:mm") + " y las " + hm2.ToString("HH:mm") + ".";
                cont = datos.IndexOf(";") + 29;
                for (int i = 0; i < 4; i++)
                {
                    string muestra = datos.Substring(cont + (i * 14), 10);
                    mostrarMuestras(muestra);
                }
                cont = datos.IndexOf(";");
                DialogResult = DialogResult.OK;
            }
            catch
            {
                MessageBox.Show("EL fichero no contiene una trama con formato
SAICA.");
            }
        }

        /*** Descodificamos los datos de cabecera de la trama SAICA ***/

        public void leerTramaSaica()
        {
            estacionOrigen = Convert.ToInt32(tramaSaica.Substring(1, 4), 16);
            estacionDestino = Convert.ToInt32(tramaSaica.Substring(5, 4), 16);
            classType = tramaSaica.Substring(9, 4);
            longDatos = Convert.ToInt32(tramaSaica.Substring(13, 4), 16);
```

```
            numItems = Convert.ToInt32(tramaSaica.Substring(17, 4), 16);
            diaSaica = Convert.ToInt32(tramaSaica.Substring(21, 4), 16);
        }

        /*** Descodificamos la fecha y hora de la trama Saica ***/

        public void convertirDiaHoraSaica()
        {
            dt = dt.AddDays(diaSaica - 1);
            horaSaica = Convert.ToInt32(tramaSaica.Substring(25, 2), 16);
            hora = horaSaica / 4;
            hm1 = hm1.AddHours(hora);
            resto = horaSaica % 4;
            minuto = 0;
            for (int i = 0; i < resto; i++)
                minuto = minuto + 15;
            hm1 = hm1.AddMinutes(minuto);
            if (minuto == 45)
                hm2 = hm2.AddHours(hm1.Hour + 1);
            else
            {
                hm2 = hm2.AddHours(hm1.Hour);
                hm2 = hm2.AddMinutes(minuto + 15);
            }
        }

        /*** Muestra los datos de cada parametros del agua ***/

        private void mostrarMuestras(string mues)
        {
            switch (mues.Substring(0, 4))
            {
                case "0000":
                    labWriteTemp.Text = string.Format("{0:D}",
(Convert.ToDecimal(int.Parse(mues.Substring(6, 4),
System.Globalization.NumberStyles.HexNumber)) / 10).ToString()) + " ºC";
                    break;
                case "0001":
                    labWriteAci.Text = int.Parse(mues.Substring(6, 4),
System.Globalization.NumberStyles.HexNumber).ToString() + " pH";
                    break;
                case "0002":
                    labWriteCon.Text = int.Parse(mues.Substring(6, 4),
System.Globalization.NumberStyles.HexNumber).ToString() + " \u00B5S/cm";
                    break;
                case "0003":
                    labWriteOxi.Text = int.Parse(mues.Substring(6, 4),
System.Globalization.NumberStyles.HexNumber).ToString() + " ppm";
                    break;
                default:
                    break;
            }
        }

        /*** Abre una ventana emergente para guardar los datos del fichero con
otra estructura ***/

        public void butguar_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Txt File|*.txt";
            saveFileDialog.Title = "Guarda las muestras";
```

```
            saveFileDialog.FileName = 'E' + estacionOrigen.ToString() + " dia " +
dt.ToString("dd-MM-yyyy") + " hora " + hm1.ToString("HH-mm") + "_" +
hm2.ToString("HH-mm") + ".txt";
            saveFileDialog.ShowDialog();

            if (saveFileDialog.FileName != "")
            {
                FileStream fsOut = File.Open(saveFileDialog.FileName,
FileMode.OpenOrCreate, FileAccess.Write);
                StreamWriter sw = new StreamWriter(fsOut);
                sw.WriteLine("Estación Origen: " + estacionOrigen.ToString());
                sw.WriteLine("Estación Destino: " + estacionDestino.ToString());
                sw.WriteLine("Clase y tipo de mensaje: " + classType);
                sw.WriteLine("Longitud de datos: " + longDatos.ToString());
                sw.WriteLine("Numero de items: " + numItems.ToString());
                sw.WriteLine("Dia Saica: " + String.Format("{0:dd/MM/yyyy}",
dt));
                for (int i = 0; i < (numItems - 1); i++)
                {
                    sw.WriteLine("");
                    sw.WriteLine("QUINCEMINUTO: " +
Convert.ToInt32(tramaSaica.Substring(25 + (i * 14), 2), 16).ToString());
                    sw.WriteLine("MEDIOMINUTO: " +
Convert.ToInt32(tramaSaica.Substring(27 + (i * 14), 2), 16).ToString());
                    switch (Convert.ToInt32(tramaSaica.Substring(29 + (i * 14),
4), 16))
                    {
                        case 0000:
                            sw.WriteLine("CLASE TIPO: TEMPERATURA");
                            break;
                        case 0001:
                            sw.WriteLine("CLASE TIPO: ACIDEZ DEL AGUA");
                            break;
                        case 0002:
                            sw.WriteLine("CLASE TIPO: CONDUCTIVIDAD");
                            break;
                        case 0003:
                            sw.WriteLine("CLASE TIPO: OXIGENO DISUELTO");
                            break;
                        default:
                            sw.WriteLine("CLASE TIPO: TIPO DESCONOCIDO");
                            break;
                    }
                    sw.WriteLine("ZONA NUM: " +
Convert.ToInt32(tramaSaica.Substring(33 + (i * 14), 2), 16).ToString());
                    sw.WriteLine("VALOR " + (i + 1).ToString() + ": " +
Convert.ToInt32(tramaSaica.Substring(35 + (i * 14), 4), 16).ToString());
                }
                sw.Close();
                fsOut.Close();
            }
        }

        private void butCerrar_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Clase AcercaDe.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SAICA
{
    public partial class AcercaDe : Form
    {
        public AcercaDe()
        {
            InitializeComponent();
        }
    }
}
```